МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

СИМВОЛЬНЫЕ МАССИВЫ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C/C++

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве методических указаний для обучающихся по основным образовательным программам высшего образования по направлениям подготовки 12.03.04 Биотехнические системы и технологии, 12.03.05 Лазерная техника и лазерные технологии, 11.03.03 Конструирование и технология электронных средств, 12.03.03 Фотоника и оптоинформатика, 11.03.04 Электроника и наноэлектроника и специальности 11.05.01 Радиоэлектронные системы и комплексы

Составитель: *А. А. Меньшикова*

САМАРА
Издательство Самарского университета
2025

УДК 004.438(075)
ББК 3973.2я7
С370

Составитель ***А. А. Меньшикова***

Рецензент канд. техн. наук, доц. Л. С. Зеленко

С370 **Символьные массивы в языке программирования С/С++:** методические указания / сост. *А. А. Меньшикова;* Министерство науки и высшего образования Российской Федерации, Самарский университет. – Самара: Издательство Самарского университета, 2025. – 1 CD-ROM (2,05 Мб). – Загл. с титул. экрана. – Текст: электронный.

В методических указаниях приводятся сведения о работе с символьными массивами на языке программирования С/С++.

Методические указания предназначены для выполнения лабораторных работ по курсу «Информатика и программирование» для студентов всех специальностей, разработаны на кафедре суперкомпьютеров и общей информатики.

УДК 004.438(075)
ББК 3973.2я7

Минимальные системные требования:
PC, процессор Pentium, 160 МГц;
Microsoft Windows XP; мышь;
дисковод CD-ROM; Adobe Acrobat Reader.

Подписано для тиражирования 17.09.2025.
Объем издания 2,05 Мб.
Количество носителей 1 диск.
Тираж 11 дисков.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, Самара, Московское шоссе, 34.

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время язык программирования C++ является одним из самых распространенных среди профессиональных программистов. Предшественником языка C++ был язык C, разработанный в начале 1970-х годов. В 1979 г. на основе языка C был разработан язык C++ [2]. Главным нововведением было включение в язык понятия класса, с помощью которого была реализована концепция объектно-ориентированного программирования, облегчающая создание программ большого объема.

Подавляющее большинство пользовательских интерфейсов основано на обмене текстовыми данными с пользователем. Это относится даже к графическим пользовательским интерфейсам: всё равно много информации представлено в текстовом виде. Пользователь читает сообщения, нажимает на кнопки, на которых также зачастую также имеются надписи. С самого начала развития вычислительной техники придумывались способы хранения и обработки текстов в памяти компьютера.

Данные методические указания содержат теорию и практические примеры для работы с текстовой информацией в языке программирования C/C++. Предназначены для студентов первого курса, изучающих основы программирования.

1. СИМВОЛЬНЫЕ МАССИВЫ

Минимальная единица любого текста – это символ.

Символ – это информация, соответствующая графической единице письменной речи.

Буквы, цифры, знаки препинания, другие графические обозначения являются символами. Символы используются для взаимодействия с пользователем на понятном ему языке.

1.1. Таблицы кодировки

Символы хранятся в памяти компьютера, как обычные целые числа.

Эти целые числа, которые хранятся в памяти компьютера и обозначают определённые символы, называются **кодами символов**.

Отображение, связывающее коды символов с их графическим представлением, называется **кодировкой.** Иначе говоря, **таблица кодировки – это таблица соответствия символа и его кода.**

Например, код буквы A – 65 (в данный момент рассматриваются символы, в которых на каждый символ отводится 1 байт).

Современным стандартом такой однобайтовой кодировки является таблица **ASCII** (American standard code for information interchange), разработанная в 1967 году Американским национальным институтом стандартов (ANSI).

В такой кодировке можно использовать всего 2^8=256 различных символов.

Кодировка ASCII содержит: десятичные цифры, латинский алфавит, национальный алфавит, знаки препинания, управляющие символы.

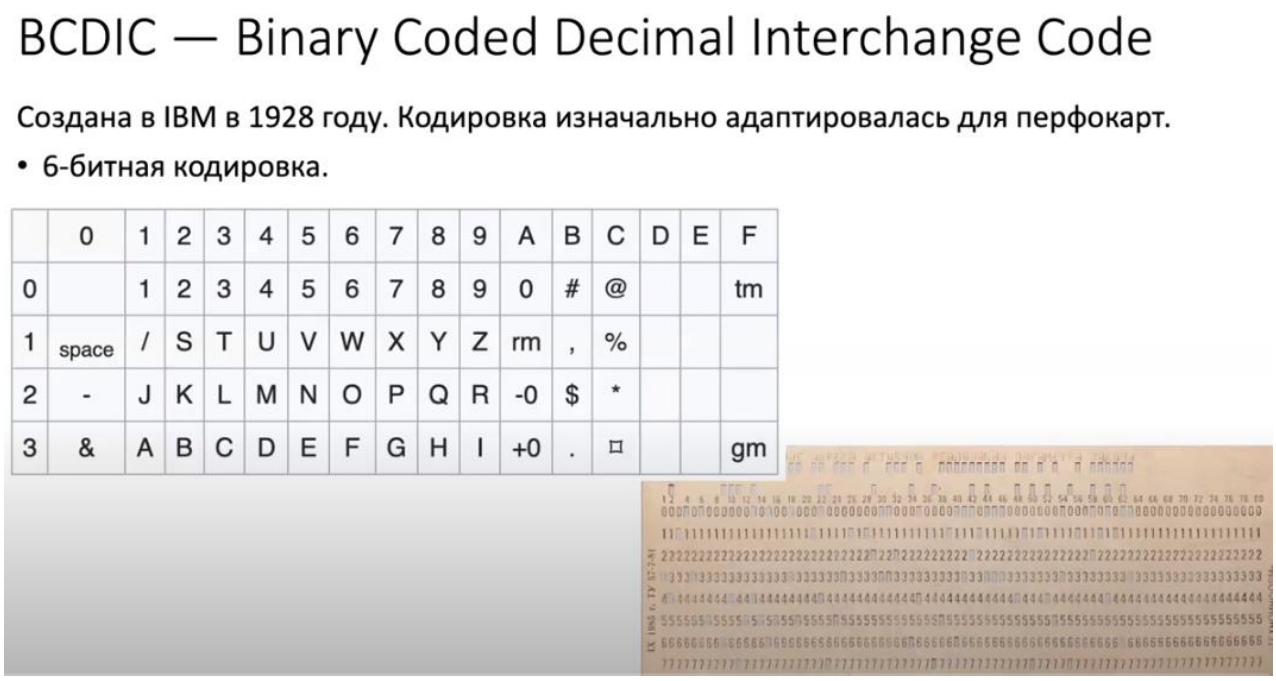

Рис. 1. Таблица кодов BCDIC

Первые 32 символа в таблице ASCII не имеют графических представлений, а предназначены для передачи служебной информации о тексте. Такие символы называются управляющими символами. В си-подобных языках для обозначения этих символов используются управляющие последовательности, начинающиеся с символа «\». Примеры часто используемых управляющих последовательностей: «\n» – перевод строки, «\r» – возврат каретки, «\t» – символ табуляции, «\0» – нулевой символ.

На рисунке 2 видно, что таблица ASCII содержит 128 символов, однако в одном байте можно закодировать 256 символов. Большая часть кодировок, кодирующих каждый символ одним байтом, основаны на этом факте: первые 128 символов в них соответствуют таблице ASCII, а остальные символы отличаются, например, для возможности добавлять коды национальных алфавитов. Так устроены, например, кодировки **Windows1251, KOI8-R, MacCyrillic** и многие другие. Добавление дополнительной кодовой страницы порождало множество проблем, например:

- дополнительные настройки операционной системы;
- при хранении текстовой информации необходимо было хранить ее кодовую страницу, а при смене кодировки (что само по себе неудобно), можно было потерять часть текста.

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	

Рис. 2. Таблица ASCII

Рис. 3. Примеры кодовых страниц

В конце 20 века началась разработка 2-байтовой кодировки Unicode, и постепенно операционные системы и программное обеспечение перешло на нее. Разработка Unicode продолжается и в настоящее время (в данный момент кодировка содержит более 2 млн. символов).

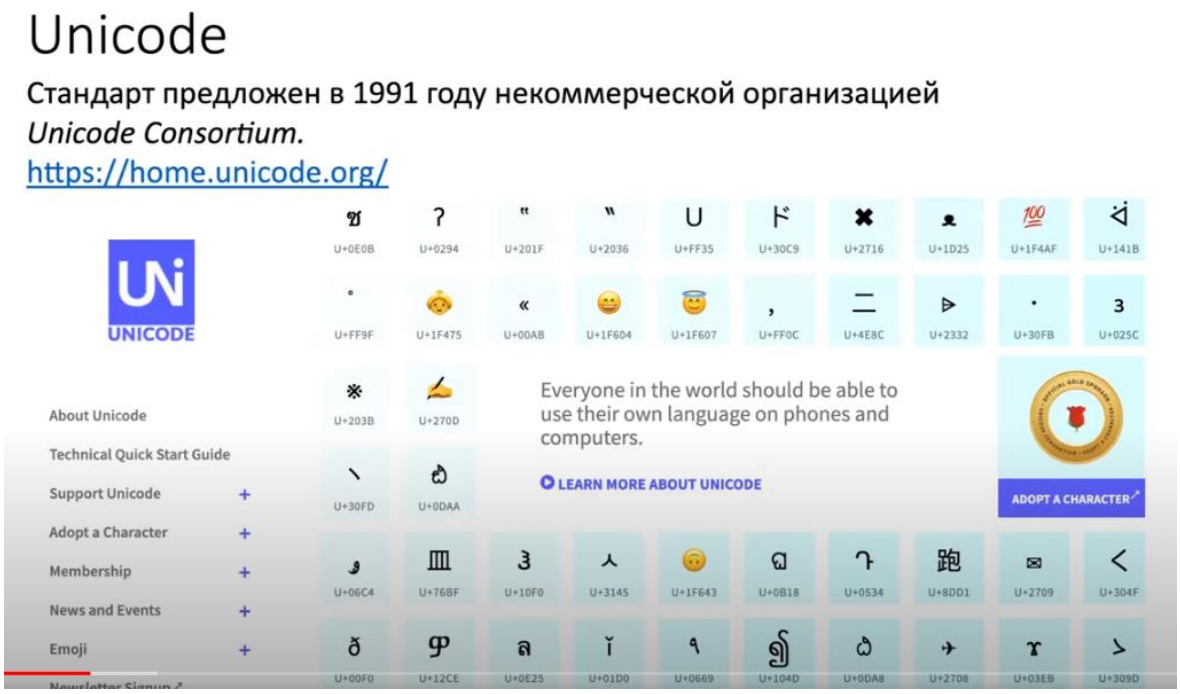

Рис. 4. Сайт UNICODE

1.2. Хранение строк в памяти компьютера

Понятно, что символьная строка – это последовательность символов. Каждому символу в памяти компьютера соответствует свой числовой код, который хранится в двоичной системе счисления.

Строка, как и другие переменные, записывается в память, причем компьютеру все равно, какие данные записаны – для него это набор байтов. Как же определить, где заканчивается строка? В числовом массиве в любой момент обращения к нему, должен быть известен его размер. С символьными массивами работать так не очень удобно в силу особенностей работы со строками (с текстом выполняются другие действия, нежели с числами). Поэтому, объявляя строку определенного размера, нужно понимать, что истинные значения данной строки могут отличаться по длине при каждом запуске программы (например, пользователь должен ввести ФИО, которое может иметь разный размер). Тем не менее, нужна возможность в любой момент определить истинную длину строки. Для решения этой проблемы можно использовать два способа:

- хранить длину строки в отдельной ячейке (как, например, в языке Pascal);
- выбрать один особый символ, который будет обозначать конец строки, причем в середине строки этот символ не может встречаться.

В языке C/C++ принят второй подход.

Символьная строка – это последовательность символов, которая заканчивается символом с кодом 0.

Символ с кодом ноль не имеет никакого изображения, в программе его записывают как **'\0'.**

Символ с кодом ноль (обозначается как **'\0'**) и цифра ноль (обозначается **'0'**, имеет код 48) – это два разных символа.

1.3. Объявление и инициализация строк

Строка представляет собой массив символов, поэтому и объявляется она именно как массив [2]:

char s[80];

Однако строка отличается от числового массива тем, что она заканчивается символом с кодом 0 – признаком окончания строки, поэтому: **если массив символов будет использоваться как строка, надо выделять на 1 байт больше памяти.**

При выделении памяти глобальные переменные заполняются нулями, а локальные содержат «мусор». Начальное значение строки можно задать при объявлении в двойных кавычках после знака равенства:

char s[80] = "Привет, Вася!";

Символы в кавычках будут записаны в начало массива **s**, а затем – признак окончания строки **'\0'**. Оставшиеся символы не меняются, и в локальных строках там будет «мусор».

Можно написать и так

char s[] = "Привет, Вася!";

В этом случае компилятор подсчитает символы в кавычках, выделит памяти на 1 байт больше и занесет в эту область саму строку и завершающий ноль.

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

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

char s1[7] = {‘m’, ‘o’, ‘o’, ‘n’, ‘\0’);

После такого объявления массивы символов s1 и s2 будут иметь вид:

s1:	m	o	o	n	\0		
s2:	m	o	o	n	\0	\0	\0

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

char s1[5] = "moon";

char s1[7] = "moon";

Заметим, что после объявления строки, например,

char s[10];

абсолютно недопустима запись вида

s = “Hello”;

т.е. имя массива, как указатель, нельзя использовать в левой части операции присваивания.

Имя строки является указателем на первый ее символ. Т.е. со строкой можно обращаться как с указателем, например, передавать параметр в функции вида char *s.

В более ранних версиях языка работала конструкция

char *s = "Hello, World";

Однако, сейчас это считается устаревшим синтаксисом и не проходит компиляцию.

Рабочий код:

char s[] = "Привет, Вася!";

char *ps = s;

Если строка не будет изменяться во время работы программы, то можно объявить константу (постоянную строку) так:

const char PRIVET[] = "Привет, Вася!";

1.4. Функции проверки символа

Данная группа функций осуществляет обработку отдельного символа, передаваемого в качестве входного параметра.

Таблица 1. Функции обработки символа

Прототип	Описание функции
int isdigit(int c)	Проверка, является ли символ с десятичной цифрой
int islower(int c)	Проверка, является ли символ с латинской буквой нижнего регистра ‘a’-‘z’
int isupper(int c)	Проверка, является ли символ с латинской буквой верхнего регистра ‘A’-‘Z’
int isalpha(int c)	Проверка, является ли символ с латинской буквой (isalpha(c) = islower(c) || isupper(c))
int isalnum(int c)	Буква либо цифра (isalnum(c) = isalpha(c) || isdigit(c))
int tolower(int c)	Перевод латинского символа с на нижний регистр
int toupper(int c)	Перевод латинского символа с на верхний регистр
int isspace(int c)	Проверка, является ли символ с пробелом (' '), сменой страницы ('\f'), новой строкой ('\n'), возвратом каретки ('\r'), табуляцией ('\t'), вертикальной табуляцией ('\v')
int iscntrl(int c)	Проверка, является ли символ с управляющим символом
int isprint(int c)	Проверка, является ли символ с печатаемым символом
int isgraph(int c)	Проверка, является ли символ с печатаемым символом, включая пробел

Пример 1.

```
cout << '5' << (isdigit('5')? " is ": " not is ") << "a digit ";
cout << 'w' << (isdigit('w')? " is ": " not is ") << "a digit ";
```

```
5 is a digit
w not is a digit
```

Рис. 5. Экран выполнения кода примера 1

1.5. Ввод и вывод строк

Логично, что в начале изучения нового объекта (в данном случае строки), необходимо рассмотреть возможности ввода и вывода таких данных. Рассмотрим способы ввода символьного массива с клавиатуры и вывода его на экран.

Все примеры ниже выполнены в интегрированной среде разработки Microsoft Visual Studio 2022 Express Edition, тип приложения – консольный.

Пример 2.

Для ввода/вывода строки можно использовать универсальный потоковый механизм cin/cout.

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char Name[50];
	cout << "Как тебя зовут? ";
	cin >> Name;
	cout << "Привет!" << Name;
}
```

Обратите внимание на функции установки кодировки SetConsoleCP(1251) и SetConsoleOutputCP(1251). Это необходимо для обеспечения возможности ввода текста на русском языке (просто setlocale(LC_ALL, “Russian”) недостаточно, т.к. эта функция обеспечивает только ВЫВОД в кириллице).

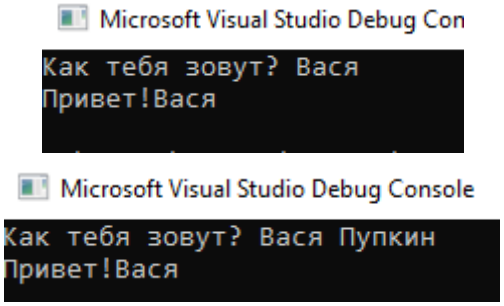

Рис. 6. Экраны выполнения кода примера 2

В данном примере можно наблюдать, что потоковый ввод строки через cin обеспечивает чтение только одного ТОКЕНА (в данном случае одного слова до пробела). Если нужно считывать предложение из нескольких слов, необходимо пользоваться другими способами.

Пример 3.

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char Name[50];
	cout << "Как тебя зовут? ";
	gets_s(Name);
	cout << "Привет!" << Name << endl;
}
```

Примечание: в старых версиях возможно изменение вызова функции gets_s() на gets(). Например, в онлайн компиляторе onlinegdb.com листинг выглядит так:

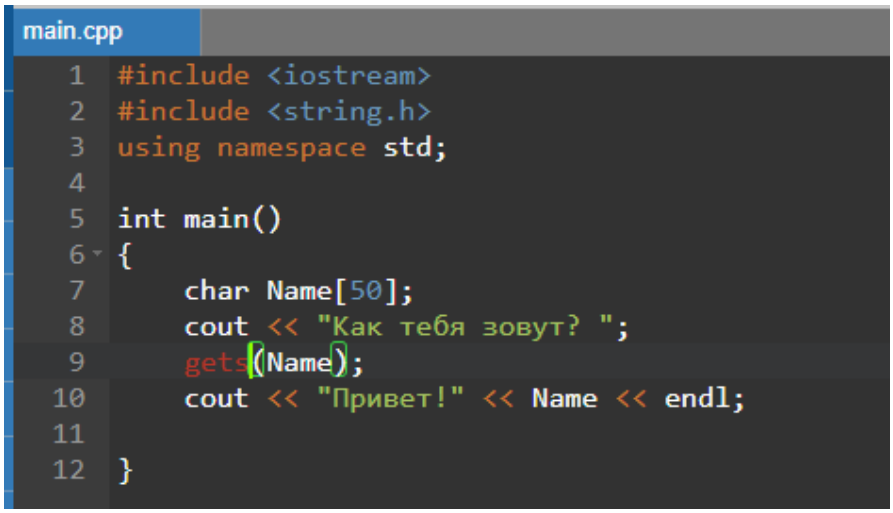

Рис. 7. Код примера 3 в онлайн-компиляторе

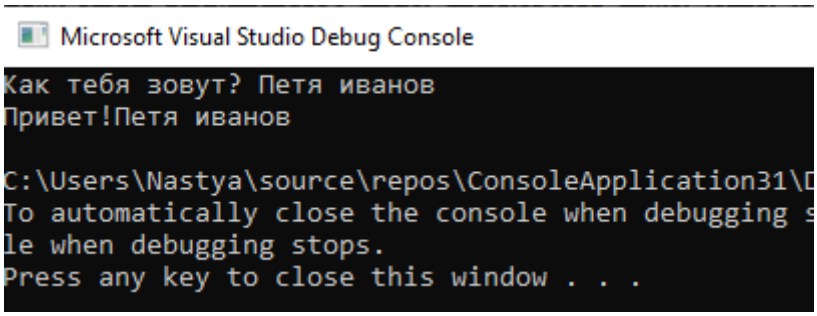

Рис. 8. Экран выполнения кода примера 3

Более современным способом считывания строки в консоли является использование функции getline.

Пример 4.

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char Name[50];
	cout << "Как тебя зовут? ";
	cin.getline(Name, sizeof(Name));
	cout << "Привет!" << Name << endl;
}
```

Второй параметр этой функции – размер строки, можно поставить числом (например, 50) или автоматически определить размер через функцию sizeof().

Пример 5 (без необходимости использования русского алфавита).

```
#include <iostream>
using namespace std;
int main() {
	char Name[50];
	cout << "What's your name? ";
	cin.getline(Name, sizeof(Name));
	cout << "Hello," << Name << endl;
}
```

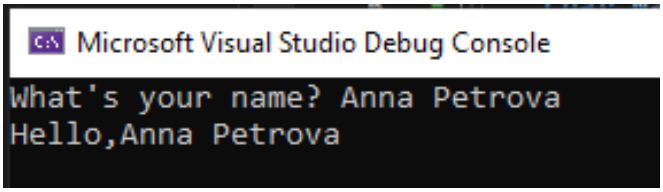

Рис. 9. Экран выполнения кода примера 5

Рассмотрим примеры с посимвольной обработкой строк.

Пример 6. Ввести символьную строку и заменить в ней все буквы 'А' на буквы 'Б'.

Будем рассматривать строку как массив символов. Надо перебрать все элементы массива, пока мы не встретим символ **'\0'** (признак окончания строки) и, если очередной символ – это буква **'А'**, заменить его на **'Б'**. Для этого используем цикл **while**, потому что мы заранее не знаем длину строки. Условие цикла можно сформулировать так: «пока не конец строки».

```
#include <iostream>
using namespace std;
int main() {
	char s[50];
	int i = 0;
	cout << "Input string: " << endl;
	cin.getline(s, sizeof(s));
	while (s[i] != '\0') {
		if (s[i] == 'A') s[i] = 'B';
		i++;
	}
	cout << "Rezultat: " << s << endl;
}
```

Цикл while можно заменить на for

```
for (i = 0; s[i]; i++)
	if (s[i] == 'A') s[i] = 'B';
```

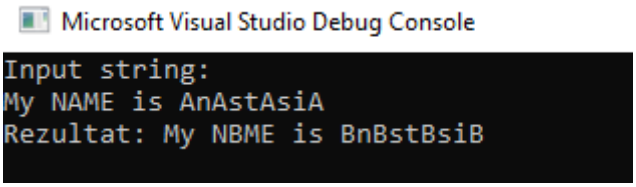

Рис. 10. Экран выполнения кода примера 6

Заметьте, что одиночный символ записывается в апострофах, а символьная строка – в кавычках.

Пример 7. Удалить из строки символ «круглая скобка».

Один из способов удаления символа из строки – переписать все символы в новую строку, исключая ненужные.

```
#include <iostream>
using namespace std;
int main(){
	char s[] = "((sss(((fffggg((";
	char d[sizeof(s)];
	int i = 0, j = 0;
	while (s[i] != '\0') {
		if (s[i] != '(') { d[j] = s[i]; j++; }
		i++;
	}
	d[j] = '\0';
	cout << d << endl;
}
```

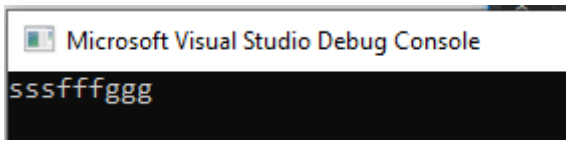

Рис. 11. Экран выполнения кода примера 7

Более оптимальный способ – удаление символов без создания новой строки. Для этого сначала продвинемся по строке до первого удаляемого символа. Затем, двигаясь по строке дальше, в случае встречи с нужным символом, будем переписывать его на место предыдущего. В последний символ запишем конец строки (‘\0’).

Сделаем наш алгоритм универсальным, оформив удаление нужного символа в виде функции.

Пример 7. (Модифицированный)

```
#include <iostream>
using namespace std;
void DeleteSymbol(char* s, const int c) {//функция удаления из строки s символов c
	int i, j;
```

```
    for (j = 0; s[j] && s[j] != c; j++);//идем до первого символа c
    for (i = j; s[i]; i++) //сдвигаем символ на место предыдущего, если необходимо
        if (s[i] != c)
            s[j++] = s[i];
    s[j] = '\0'; //«закрываем» строку
}
int main() {
    char s[] = "((sss(((fffggg((";
    DeleteSymbol(s, 's');
    cout << s << endl;
    DeleteSymbol(s, '(');
    cout << s << endl;
}
```

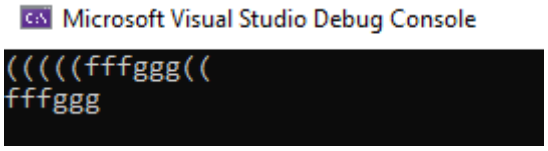

Рис. 12. Экран выполнения кода модифицированного примера 7

1.6. Лабораторная работа № 1. Обработка строк посимвольно

Выполнить обработку строками ASCII, тип данных char[], соблюдая следующие требования:

- все действия выполняются без использования библиотечных функций;
- исходные строки не содержат букв русского алфавита;
- вход в программу осуществляется с использованием символьного пароля (реализовать в виде функции);
- основной алгоритм реализовать в виде функций.

Уровень сложности 1

1. Ввести с экрана строку. В строке подсчитать количество предложений. Вывести на экран исходную строку и количество предложений в ней с соответствующими комментариями. Предложения в строке разделяются знаками «.», «!», «?».

2. Ввести с экрана строку. Вывести на экран исходную строку, а также отдельно первое и последнее слова с соответствующими комментариями. Слова разделяются пробелами.

3. Ввести с экрана строку. В строке подсчитать количество цифр. Вывести на экран исходную строку и количество цифр в ней с соответствующими комментариями.

4. Ввести с экрана строку. В строке заменить все буквы «c» на «k», «w» на «v» и подсчитать количество букв «k» и «v».

5. Ввести с экрана строку. Заменить в строке все символы на «1», а пробелы на «0». Подсчитать нули.

6. Ввести с экрана строку с арифметическим выражением, обязательно содержащим цифры 1 и 5. В строке заменить все цифры 1 на I, цифры 5 на V и подсчитать количество символов «=».

7. Ввести с экрана строку, состоящую из букв, цифр и знаков препинания. В строке подсчитать количество букв.

8. Ввести с экрана строку. Определить, какой символ встречается в ней раньше – ‘x’ или ‘w’. Если какого-то из символов нет, вывести сообщение об этом.

9. Ввести с экрана две строки. Вывести бОльшую по длине строку столько раз, на сколько символов отличаются строки.

10. Ввести с экрана строку. Найти сумму имеющихся в ней цифр.

11. Вывести исходную строку наоборот.

12. Вывести на экран все цифры, строчные и прописные латинские буквы с их кодами.

13. Дана строка S размером n. Сформировать и вывести строку длиной 2n, которая состоит из символов строки S, между которыми вставлено по 1 пробелу.

14. Дано целое число n в интервале от 1 до 26. Сформировать строку из n последних строчных латинских букв в обратном порядке, начиная с ‘z’.

Уровень сложности 2

1. Ввести с экрана строку. В строке подсчитать количество знаков препинания (запятая, точка, восклицательный знак). Затем удалить из строки знаки препинания.

2. Дана строка S. Определить, встречаются ли в ней 2 идущих подряд символа. Вывести такие символы и их позиции.

3. Подсчитать в строке S частоту вхождения каждого символа.

4. Дана строка, изображающая двоичную запись целого положительного числа. Получить строку, изображающую десятичную запись этого числа.

5. Дана строка, изображающая десятичную запись целого положительного числа. Получить строку, изображающую двоичную запись этого числа.

6. Заменить в строке каждую серию подряд идущих пробелов на один пробел.

7. Найти в строке все целые неотрицательные числа и вывести их на экран.

8. Пусть S1, S2 – некоторые строки. Найти количество символов (без учета ‘\0’), которые присутствуют: одновременно в строках S1 и S2; хотя бы в одной из S1 и S2; ни в одной из строк S1 и S2.

9. Удалить из строки символы, расположенные между скобками ‘(‘ ‘)’. Предполагается, что вложенных скобок нет.

10. Подсчитать в строке гласные и согласные латинские буквы.

11. Удалить из строки S все символы строки S1.

2. ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ

В языке C/C++ имеется достаточно много специальных функций, которые работают со строками – последовательностями символом с нулевым символом на конце [3]. Для использования этих функций надо включить в программу заголовочный файл:

#include <string>

В таблице приведены основные функции операций над строками (библиотека <string>).

Таблица 2. Функции работы со строками типа char[]

Прототип	Описание функции
int strlen(const char *s)	Возвращает длину строки
char *strcpy(char *s, const char *t)	Копирует строку t в строку s, включая ‘\0’; возвращает s
char *strcpy(char *s, const char *t, size_t n)	Копирует не более n символов строки t в s; возвращает s. Дополняет результат символами ‘\0’, если символов в t меньше значения n
char *strcat(char *s, const char *t)	Присоединяет t к s; возвращает s
char *strncat(char *s, const char *t, size_t n)	Присоединяет не более n символов t к s; возвращает s
int strcmp(const char *s, const char *t)	Сравнивает t и s; возвращает <0, если s<t, 0, если s == t, и >0, если s>t
int strncmp(const char *s, const char *t, size_t n)	Аналогична функции strcmp(), только сравниваются не более n первых символов в строках s и t
char *strchr(const char *s, int c)	Возвращает указатель на первое вхождение символа c в строку s либо NULL, если такого символа не оказалось
char *strrchr(const char *s, int c)	Возвращает указатель на последнее вхождение символа c в строку s, либо NULL, если такого символа не оказалось
char *strpbrk(const char *s, const char *t)	Возвращает указатель в строке s на первый символ, который совпал с одним из символов, входящих в t либо NULL, если такового не оказалось
char *strstr(const char *s, const char *t)	Возвращает указатель на первое вхождение строки t в строку s либо NULL, если такой подстроки в s не оказалось
char *strtok(char *s, const char *t)	Ищет в строке s лексему, ограниченную символами из t. Возвращает указатель на первый символ лексемы либо NULL, если лексемы не существует

Многие из этих функций достаточно опасны при неправильном использовании, потому что они не проверяют, достаточно ли выделено памяти для копирования, перемещения или другой операции, единственным признаком окончания строки для них является символ **'\0'.**

2.1. Длина строки

Это самая простая функция, которая определяет, сколько символов в переданной ей строке (не считая '**\0**'). Ее имя происходит от английских слов ***str****ing* ***len****gth* (длина строки).

Пример 8. Использование функции strlen.

```
#include <iostream>
using namespace std;
int main() {
	int len;
	char s[] = "fakultet";
	len = strlen(s);
	cout << "Length of string " << s << " = " << len << endl;
}
```

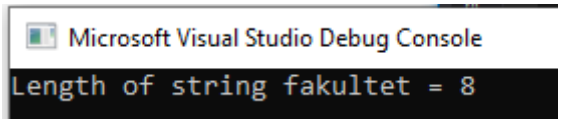

Рис. 13. Экран выполнения кода примера 8

Реализация функции подсчета символов может выглядеть следующим образом:

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

2.2. Сравнение строк

Для сравнения двух строк используют функцию **strcmp** (от английских слов ***str****ing* ***comp****arison* – сравнение строк). Функция возвращает ноль, если строки равны (то есть «разность» между ними равна нулю) и ненулевое значение, если строки различны. Сравнение

происходит по кодам символов, поэтому функция различает строчные и заглавные буквы – они имеют разные коды.

Пример 9. Использование функции strcmp.

```
char s1[] = "Vasya";
char s2[] = "Petya";
if (0 == strcmp(s1, s2))
        cout << "Strings "<< s1 << " and " << s2 << " are same" << endl;
else
        cout << "Strings " << s1 << " and " << s2 << " are different" << endl;
```

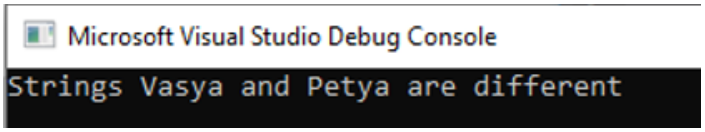

Рис. 14. Экран выполнения кода примера 9

Если строки не равны, функция возвращает «разность» между первой и второй строкой, то есть *разность кодов первых различных символов*. Эти числа можно использовать для сортировки строк – если «разность» отрицательна, значит первая строка «меньше» второй, то есть стоит за ней в алфавитном порядке. В таблице показано несколько примеров (код буквы **'A'** равен **65**, код буквы **'B'** – **66**, код буквы **'C'** – **67**).

Таблица 3.Функция strcmp

s1	s2	strcmp(s1, s2)
AA	AA	0
AB	AAB	‘B’ – ‘A’ = 66 – 65 = 1
AB	CAA	‘A’ – ‘C’ = 65 – 67 = -2
AA	AAA	‘\0’ – ‘A’ = -65

Пример 10. Ввести две строки и вывести их в алфавитном порядке.

```
#include <iostream>
using namespace std;
int main() {
        char s1[80], s2[80];
        cout << "Input string 1: ";
        cin.getline(s1, sizeof(s1));
        cout << "Input string 2: ";
        cin.getline(s2, sizeof(s2));
        if (strcmp(s1, s2) <= 0)
                cout << s1 << endl << s2 << endl;
        else
                cout << s2 << endl << s1 << endl;
}
```

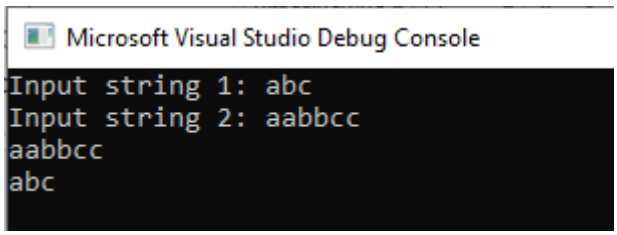

Рис. 15. Экран выполнения кода примера 10

Иногда надо сравнить не всю строку, а только первые несколько символов. Для этого служит функция **strncmp** (с буквой **n** в середине). Третий параметр этой функции – количество сравниваемых символов. Принцип работы такой же – она возвращает нуль, если заданное количество первых символов обеих строк одинаково.

Пример 11. Ввести две строки и сравнить первые два символа.

```
char s1[80], s2[80];
cout << "Input string 1: ";
cin.getline(s1, sizeof(s1));
cout << "Input string 2: ";
cin.getline(s2, sizeof(s2));
if (0 == strncmp(s1, s2, 2))
        cout << "Pervye dva simvola " << s1 << " i " << s2 << " odinakovy" << endl;
else
        cout << "Pervye dva simvola " << s1 << " i " << s2 << " raznye" << endl;
```

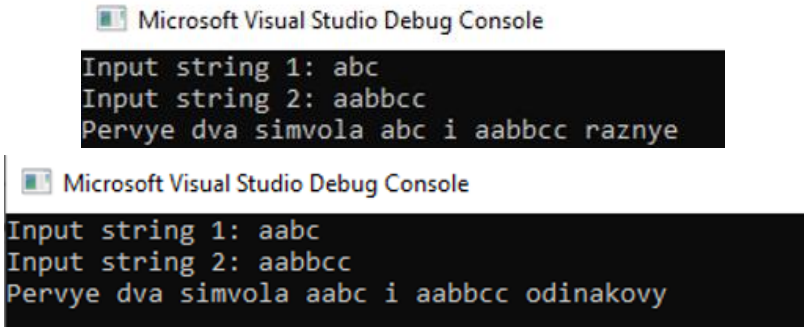

Рис. 16. Экраны выполнения кода примера 11

Один из примеров использования функции **strcmp** – проверка пароля. Составим программу, которая спрашивает пароль и, если пароль введен неверно, заканчивает работу, а если верно – выполняет какую-нибудь задачу.

Пример 12. Составить программу, которая определяет, сколько цифр в символьной строке. Программа должна работать только при вводе пароля «куку».

```
#include <iostream>
using namespace std;
int main() {
        char s[80], pass[] = "kuku";
```

```
    int i = 0, count = 0;
    cout << "Input password: ";
    cin.getline(s, sizeof(s));
    if (0 != strcmp(pass, s)) {
        cout << "Password is incorrect!" <<endl;
        return 1;
    }
    cout <<"Input string: ";
    cin.getline(s, sizeof(s));
    while (s[i] != '\0') {
        if (s[i] >= '0' && s[i] <= '9') count++;
        i++;
    }
    cout << count << " digits is found" << endl;
    return 0;
}
```

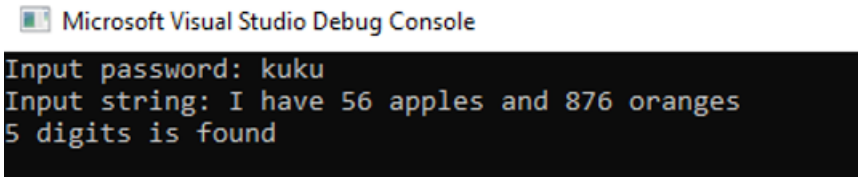

Рис. 17. Экран выполнения кода примера 12

В этой программе использован тот факт, что коды цифр расположены в таблице символов последовательно от **'0'** до **'9'**. Поэтому можно использовать двойное неравенство, а не сравнивать текущий символ **s[i]** с каждой из цифр. Обратите внимание на разницу между символами **'\0'** (символ с кодом 0, признак конца строки) и **'0'** (символ с кодом 48, цифра 0).

Пример 13. Подсчет количества пробелов (или любых других символов).

```
len = strlen(s);
for (i = 0; i < len; i++) {
    if (s[i] == ‘ ‘)
        prob++;
}
```

2.3. Копирование строк

Часто необходимо записать новое значение в строку или скопировать информацию из одной строки в другую. Функции копирования принадлежат к числу «опасных» – они могут вызвать серьезную ошибку, если произойдет **выход за границы массива**. Это бывает в том случае, если строка, в которую копируется информация, имеет недостаточный **размер** (под нее выделено мало место в памяти).

В копировании участвуют две строки, они называются «*источник*» (строка, **откуда** копируется информация) и «*приемник*» (**куда** она записывается или добавляется).

При копировании строк надо проверить, чтобы для строки-приемника было выделено достаточно места в памяти.

Простое копирование выполняет функция **strcpy**. Она принимает два аргумента: сначала строка-приемник, потом – источник (порядок важен!).

Пример 14. Использование функции strcpy.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
int main() {
	char s1[50], s2[10];
	cin.getline(s1, sizeof(s1));
	strcpy(s2, s1); // s2 (приемник) <- s1 (источник)
	cout << s2 << endl;
}
```

Этот фрагмент программы является «опасным» с точки зрения выхода за границы строки. В строку **s1** можно безопасно записать не более 49 символов (плюс завершающий ноль). Поэтому если с клавиатуры будет введена строка длиннее 49 символов, при записи ее в память произойдет выход за границы строки **s1**. Строка **s2** может принять не более 9 символов, поэтому при большем размере **s1** произойдет выход за границы строки **s2.**

Директива _CRT_SECURE_NO_WARNINGS предназначена для отключения ошибки компилятора, связанной с использованием небезопасной функции strcpy. Вместо нее предлагается использовать функцию strcpy_s. Однако в обоих случаях выполнение программы будет прервано, если строка s1 (источник) будет длиннее 9 символов.

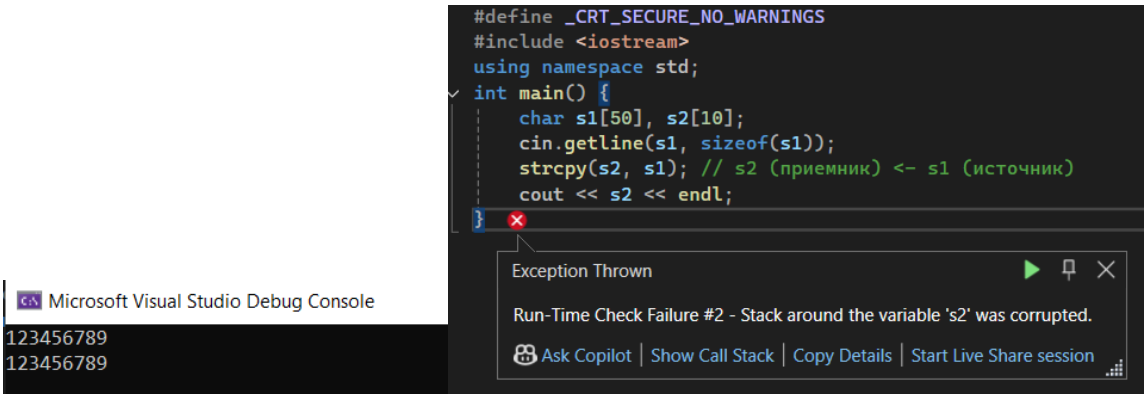

Рис. 18. Экраны успешного и неуспешного выполнения кода примера 14

Поскольку реально функции передается адрес начала строки, можно заставить функцию начать работу любого символа, а не только с начала строки. Например, следующая строка скопирует строку **s2** в область памяти строки **s1**, которая начинается с ее 6-ого символа, оставив без изменения первые пять: strcpy (s1+5, s2).

При этом надо следить, чтобы не выйти за границу массива. Кроме того, если до выполнения этой операции в строке **s1** меньше 5 символов, то программа будет прервана.

Еще одна похожая функция позволяет скопировать только заданное количество символов, она называется **strncpy** и принимает в третьем параметре количество символов, которые надо скопировать. Важно помнить, что эта функция **НЕ записывает завершающий нуль**, а только копирует символы (в отличие от нее **strcpy** всегда копирует завершающий нуль). Функция **strncpy** особенно полезна тогда, когда надо по частям собрать строку из кусочков.

Пример 15. Использование функции strncpy.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
int main() {
	char s1[] = "ku-ku", s2[10];
	strncpy(s2, s1, 2); // скопировать 2 символа из s1 в s2
	cout << s2 <<endl; // ошибка! нет последнего '\0'
	s2[2] = '\0'; // добавляем символ окончания строки
	cout << s2 << endl; // ошибка! нет последнего '\0'
}
```

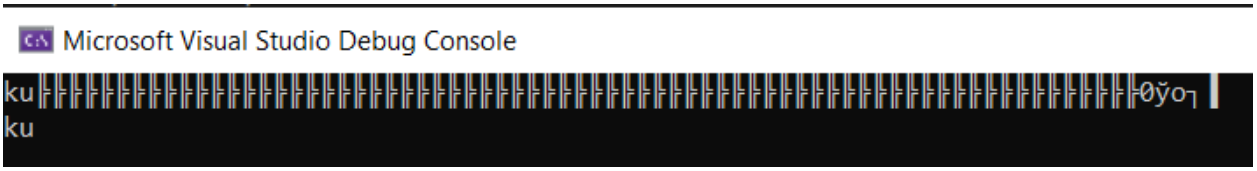

Рис. 19. Экран выполнения кода примера 15

Реализация функции копирования строки может выглядеть следующим образом:

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

2.4. Объединение строк

Еще одна функция – **strcat** (от ***str****ing con****cat****enation* – сцепка строк) позволяет добавить строку источник в конец строки-приемника (завершающий нуль записывается автоматически).

Надо только помнить, что приемник должен иметь достаточный размер, чтобы вместить обе исходных строки. Функция **strcat** автоматически добавляет в конец строки-результата завершающий символ **'\0'**.

Пример 16. Использование функции strcat.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
int main() {
```

```
    setlocale(LC_ALL, "Russian");
    char s1[80] = "Могу, ",
     s2[] = "хочу, ",
     s3[] = "надо!";
    strcat(s1, s2); // дописать s2 в конец s1
    cout << s1 << endl; // "Могу, хочу, "
    strcat(s1, s3); // дописать s3 в конец s1
    cout << s1 << endl;  // "Могу, хочу, надо!"
}
```

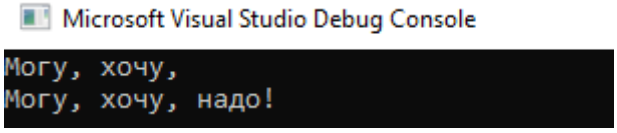

Рис. 20. Экран выполнения кода примера 16

Заметьте, что если бы строка **s1** была объявлена как **s1[]** (или с длиной меньше 18), произошел бы выход за границы массива с печальными последствиями.

Собственная реализация функции копирования строки может выглядеть следующим образом:

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

Пример 17. Ввести с клавиатуры имя файла. Изменить его расширение на ".exe".

Алгоритм решения:

1. Найти в имени файла точку **'.'** или признак конца строки **'\0'**.
2. Если нашли точку, скопировать начиная с этого места новое расширение **".exe"** (используем функцию **strcpy**).
3. Если нашли конец строки (точки нет), добавить в конец расширение **".exe"** (используем функцию **strcat**).

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
int main() {
    char s[80];
    int n; // номер символа '.'
    cout <<"Get name of file – ";
    cin.getline(s, sizeof(s));
    n = 0;
    while ((s[n] != '.') // ищем первую точку
```

```
    && (s[n] != '\0')) // или конец строки
        n++;
    if (s[n] == '.') // если нашли точку, то...
        strcpy(s + n, ".exe"); // меняем расширение,
    else
        strcat(s, ".exe"); // иначе добавляем
    cout << s <<endl;
}
```

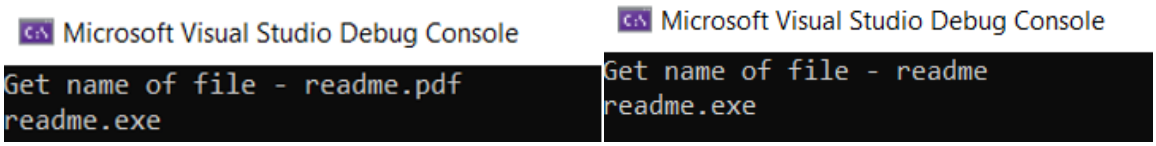

Рис. 21. Экран выполнения кода примера 17

2.5. Поиск в строках

Когда говорят о поиске в строках, обычно рассматривают две задачи: найти первый заданный символ с начала (или с конца), или также найти заданную подстроку (если она есть).

Первую задачу выполняют функции **strchr** (поиск с начала строки) и **strrchr** (поиск с конца строки), а вторую – функция **strstr.**

Все эти функции возвращают указатель на найденный символ (или на первый символ найденной подстроки). Это значит, что переменная, в которую записывается это значение, должна быть объявлена как *указатель* на символьную переменную. Вспомним, что указатель – это ячейка в памяти, в которую можно записывать *адрес* другой переменной.

Структура вызова функций такая: на первом месте – строка для поиска, на втором – что искать (один символ для функций **strchr** и **strrchr** или строка для **strstr**). Т.к. каждый символ занимает 1 байт, то чтобы получить номер символа с начала строки, надо вычесть из полученного указателя адрес начала массива. Если поиск завершился неудачно, функции возвращают **NULL.**

Пример 18. Примеры использования функций поиска в строках.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
int main() {
    setlocale(LC_ALL, "Russian");
    char s1[] = "Мама мыла раму",
    s2[] = "Война и мир",
     *p;
    p = strchr(s1, 'a'); //поиск буквы а слева
    if (p != NULL) {
        cout << "Первая буква а: номер " << p – s1 << endl;
    p = strrchr(s1, 'a'); //поиск буквы а справа
```

```
        cout << "Последняя буква а: номер " << p – s1 << endl;
    }
    p = strstr(s2, "мир"); //поиск подстроки
    if (p != NULL)
        cout << "Нашли мир в " << s2 << endl;
    else
        cout << "Нет слова мир в " << s2 << endl;
}
```

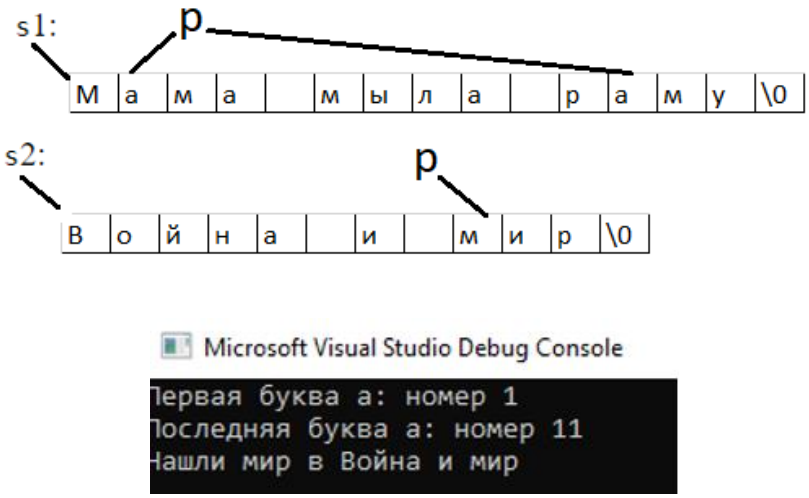

Рис. 22. Иллюстрация работы поиска в строке и экран выполнения кода примера 18

2.6. Разбор строки на лексемы

Лексе́ма (от др. -греч. λέξις — слово, выражение, оборот речи) в лингвистике – единица языка, являющаяся единицей словарного состава языка, представляющая собой совокупность всех парадигматических форм (словоформ) одного языка и их лексических значений. Проще говоря, в лингвистике лексемы – это слова и все их словоформы. Если рассматривать задачу более широко, то лексемами могут быть не только слова естественных языков, разделенные пробелами и знаками препинания, а любые наборы символов, разделенные любыми символами-разделителями. Алгоритмы разбора текста на отдельные лексемы, сохраняемые в отдельные строки, является одними из наиболее часто встречаемых задач обработки текста.

Рассмотрим один из таких алгоритмов на основе функции

char *strtok(char *string, const char *delim);

Функция strtok выполняет поиск лексем в строке string. Последовательность вызовов этой функции разбивают строку string на лексемы, которые представляют собой последовательности символов, разделенных символами разделителями.

На первый вызов, функция принимает строку string в качестве аргумента, чей первый символ используется в качестве начальной точки для поиска лексем. В последующие вызовы, функция ожидает нулевого указателя и использует позицию сразу после окончания последней лексемы как новое местонахождение для сканирования.

Для определения начала лексемы функция сначала определяет символы, не содержащиеся в строке delim, то есть они являются символами-разделителями. А затем посимвольно

проверяет остальную часть строки до первого символа-разделителя, который сигнализирует конец лексемы.

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

- **string** – строка для поиска в ней лексем. Содержание этой строки будет изменено, она разбивается на более мелкие строки (лексемы). Данный параметр может содержать нулевой указатель, в этом случае функция продолжает сканирование с того места, где был остановлен предыдущий успешный вызов функции.
- **delim – с**трока, содержащая разделители. Они могут варьироваться от одного вызова к другому вызову функции.

Директива #define DELIMITERS определяет идентификатор и последовательность символов, которые будут подставляться вместо идентификатора каждый раз, когда он встретится в исходном коде.

Пример 19. Разделить строку на лексемы с помощью функции strtok.

```
#define _CRT_SECURE_NO_WARNINGS
//отключает предупреждения о безопасности для функции strtok
#define DELIMITERS " .,:;?!-\n\t"
#include <iostream>
using namespace std;
int main() {
	setlocale(LC_ALL, "Russian");
	char str[] = "Особенности национальной  рыбалки – художественный, комедийный фильм.";
	cout << "исходная строка – " << str << endl;
	cout << "Разделение строки " << str << " на лексемы" << endl;
	char* pch = strtok(str, DELIMITERS);//" ,.-"
	while (pch != NULL) {
		cout << "pch- " << pch << " :длина слова = " << strlen(pch) << endl;
		pch = strtok(NULL, DELIMITERS);
	}
}
```

Microsoft Visual Studio Debug Console

```
исходная строка - Особенности национальной  рыбалки - художественный, комедийный фильм.
Разделение строки Особенности национальной  рыбалки - художественный, комедийный фильм. на лексемы
pch- Особенности :длина слова = 11
pch- национальной :длина слова = 12
pch- рыбалки :длина слова = 7
pch- художественный :длина слова = 14
pch- комедийный :длина слова = 10
pch- фильм :длина слова = 5
```

Рис. 23. Экран выполнения кода примера 19

Недостаток данной программы – портится исходная строка str, т.к. указатель-имя строки в процессе работы перемещается по строке.

Рассмотрим пример разделения на слова другим способом [4].

Для эффективной реализации данного алгоритма введем логический массив int flag[256], в котором порядковые номера (индексы) элементов соответствуют кодам символов из таблицы символов ASCII. Элементы данного массива инициализируем следующим образом: если символ с кодом i является символом-разделителем, то полагаем flag[i] = 1, иначе flag[i] = 0. И так для всех символов из таблицы ASCII (i = 0, 1, …, 255):

int flag[256] = {0};

for(i = 0; DELIMITERS[i]; i++)

flag[DELIMITERS[i]] = 1’;

После данного цикла массив flag будет иметь такой вид:

…	1	1	…	1	…	1	1	1	…	1	1	…	0	0	…	0	…
	9	10		32		44	45	46		58	59		65	66		90	
	\t	\n		_		,	-	.		:	;		A	B		Z	

Благодаря этому, проверка, является ли i-й символ строки text символом-разделителем, будет выглядеть следующим образом: if (flag[text[i]]).

Ниже приводится эффективный алгоритм выделения лексем из строки text. Заметим, что сложность данного алгоритма равно m+n, где m – длина строки DELIMITERS, n – длина строки text, при этом данный алгоритм работает быстрее предыдущего (через strtok).

Пример 20. Разделить строку на лексемы с помощью дополнительного массива flag.

```
#define _CRT_SECURE_NO_WARNINGS
#define DELIMITERS " .,:;?!-\n\t" //набор символов-разделителей
#include <iostream>
#include <windows.h>
using namespace std;
const int N = 1024; //размер исходной строки
typedef unsigned char UCHAR; //для использования русских букв
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    char text[N]; //исходная строка
    char* word; //адрес очередного слова в строке
    int i, j, flag[256] = { 0 }, num = 0;
    cin.getline(text, N); //вводим строку с клавиатуры
    cout << "Исходная строка – " << text << endl;
    for (i = 0; DELIMITERS[i]; i++) //если символ с кодом i является разделителем,
        flag[DELIMITERS[i]] = 1; //то присвоим flag[i] = 1
    for (i = 0; text[i] && flag[(UCHAR)text[i]]; i++);
    //пропускаем символы-разделители до первого слова в строке
    while (text[i]) {
        j = i;//позиция начала нового слова
        while (text[i] && !flag[(UCHAR)text[i]]) //идем до конца очередного слова
```

```
            i++;
            word = new char[(i – j + 1) * sizeof(char)]; //выделяем память под слово
            strncpy(word, &text[j], i – j);//копируем символы очередного слова в строку
word
            word[i – j] = '\0'; //"закрываем" наше слово
            num++; //номер слова
            cout << "Слово #"<< num << " : " << word << endl;//печатаем слово
            //////////
            //здесь делаем со словом word то, что нужно по заданию
            /////////
            delete [] word; //освобождаем динамическую память
            word = NULL;
            while (text[i] && flag[(UCHAR)text[i]])
            //пропускаем символы-разделители до следующего слова в строке
            i++;
        }
        cout << "Исходная строка – "<< text << endl;
    }
```

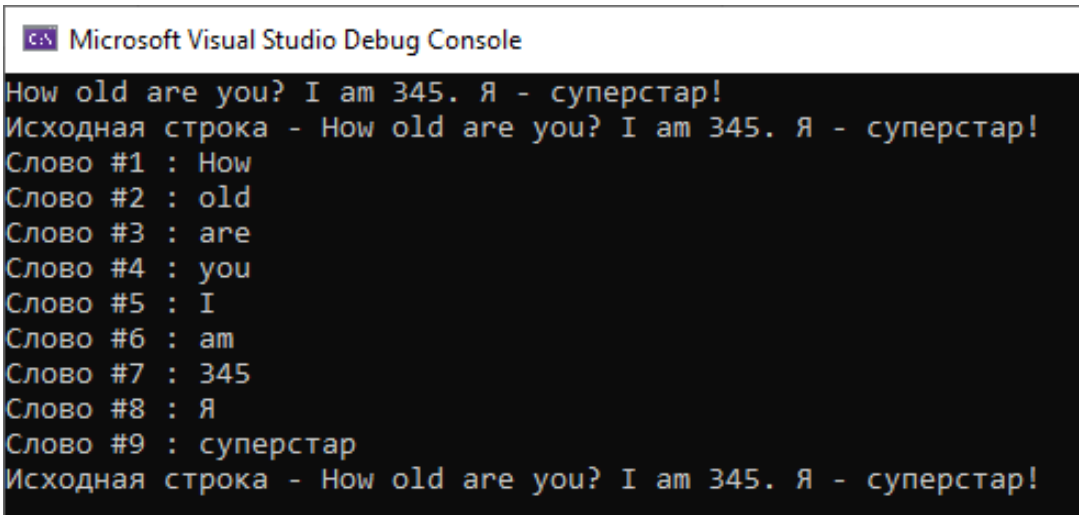

Рис. 24. Экран выполнения кода примера 20

2.7. Лабораторная работа № 2. Разбор строки на лексемы

Выполнить обработку строк ASCII, тип данных char[], соблюдая следующие требования:

- выделить в строке-предложении все слова, разделенные символами-разделителями «_.,;:\n\t!?», и обработать выделенные слова в соответствии с вариантом задания;
- за основу взять алгоритм с использованием массива flag;
- индивидуальный алгоритм, вывод заголовка, символьный пароль реализовать в виде функций;
- регулярное слово – слово, состоящее только из больших латинских букв;
- алфавитный порядок задается таблицей ASCII.

Уровень сложности 1

1. Напечатать все слова, начинающиеся на большую и заканчивающиеся на маленькую букву.

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

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

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

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

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

7. Напечатать все слова, состоящие только из цифр.

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

9. Напечатать все пятибуквенные слова.

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

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

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

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

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

15. Напечатать все слова, состоящие не более чем из пяти букв.

16. Напечатать все слова, состоящие только из латинских букв.

Уровень сложности 2

1. Подсчитать количество слов, начинающихся на большую букву и содержащих хотя бы один арифметический знак. Напечатать все слова, содержащие две рядом стоящие одинаковые буквы.

2. Подсчитать количество регулярных слов. Напечатать в перевернутом виде все слова, которые содержат два экземпляра заданного символа.

3. Напечатать все слова, начинающиеся с большой буквы. Напечатать самое длинное регулярное слово, которое состоит из одинаковых символов.

4. Напечатать слово, содержащее наибольшее количество цифр. Напечатать количество слов, содержащих хотя бы два арифметических знака.

5. Напечатать все регулярные слова. Напечатать в перевернутом виде самое длинное слово, состоящее только из цифр и латинских букв.

6. Найти количество слов, содержащих более одной цифры, и, исключив все арифметические знаки из этих слов, напечатать их. Напечатать в порядке возрастания все числа, встретившиеся в словах.

7. Определить количество слов, которые содержат заданное подслово и напечатать их. Напечатать в порядке убывания все числа, встретившиеся в словах.

8. Определить количество слов, содержащих и буквы, и цифры, и арифметические знаки. Напечатать их. Напечатать все симметричные слова.

9. Подсчитать количество регулярных слов, содержащих хотя бы две одинаковые буквы. Напечатать все слова, имеющие одну цифру, удалив из таких слов все арифметические знаки.

10. Найти самое длинное регулярное слово и удалить из него все гласные буквы. Найти все слова, в которых имеются либо только цифры, либо только латинские буквы.

11. Подсчитать количество слов, состоящих из одинаковых букв или одинаковых цифр. Напечатать в перевернутом виде слова, имеющие хотя бы один арифметический знак.

12. Напечатать все слова, которые начинаются с большой буквы и заканчиваются заданным двухбуквенным подсловом. Определить количество слов, содержащих согласные латинские буквы, и напечатать порядковые номера этих слов.

13. Напечатать все слова, имеющие в своем составе согласные латинские буквы. Определить количество слов, которые не имеют в своем составе ни одной цифры, и напечатать эти слова.

14. Напечатать все симметричные слова, предварительно удалив из них цифры. Напечатать все слова, состоящие только из согласных латинских букв.

15. Найти все слова, содержащие числа в диапазоне от 10 до 100, и подсчитать их сумму. Напечатать слова, не имеющие цифр, предварительно удалив арифметические знаки.

16. Подсчитать количество слов, начинающихся с большой буквы и оканчивающихся цифрой. Напечатать слова, содержащие заданное подслово и хотя бы один арифметический знак.

17. Подсчитать количество слов, содержащих хотя бы одну согласную латинскую букву и хотя бы одну цифру. Напечатать все слова, состоящие только из четных цифр, и подсчитать сумму этих цифр.

18. Напечатать все слова, которые содержат хотя бы один арифметический знак и заканчиваются на цифру. Определить количество слов, содержащих все маленькие латинские гласные буквы.

19. Найти количество симметричных регулярных слов и напечатать их. Напечатать в перевернутом виде все слова, содержащие согласные латинские буквы.

20. Найти и напечатать все слова, содержащие наибольшее количество букв, если только буквы расположены в алфавитном порядке. Подсчитать количество симметричных слов, имеющих более двух арифметических знаков.

3. КЛАСС STRING

В этой главе приведены некоторые сведения о классе string, определённом в языке C++. Данный тип пришел на смену char[] и имеет более удобные средства работы со строками.

Он предназначен для хранения строк, состоящих из однобайтовых символов. Тип string не хранит кодировку, так что программисту самому следует заботиться о правильном графическом отображении хранимых символов.

Рассмотрим методы работы со строками с помощью класса string.

Пример 21. Ввод строки типа string в консоли.

```
#include <iostream>
#include <string> // необходимо подключить библиотеку string
using namespace std;
int main() {
	setlocale(LC_ALL, "Russian");
	string s; //описание строки, указывать заранее размер не нужно
	cout << "Как тебя зовут? ";
	getline(cin, s);
	//ввод строки с клавиатуры, использование getline выглядит немного по другому
	cout << "Привет!" << s << endl;
}
```

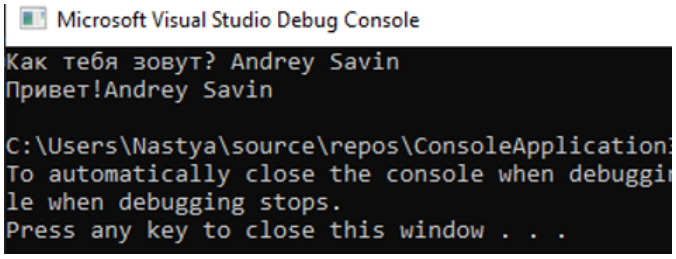

Рис. 25. Экран выполнения кода примера 21

Пример 22. Определить размер строки типа string, метод size().

```
cout << s.size() << endl;
```

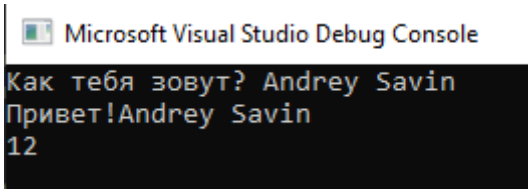

Рис. 26. Добавление в код примера 21 вызова метода определения размера строки

Пример 23. Изменение размера строки resize().

```
cout << s.size() << endl;
s.resize(10);
```

```
cout << s.size() << " – " << s << endl;
s.resize(25, 'A');
cout << s.size() << " – " << s << endl;
```

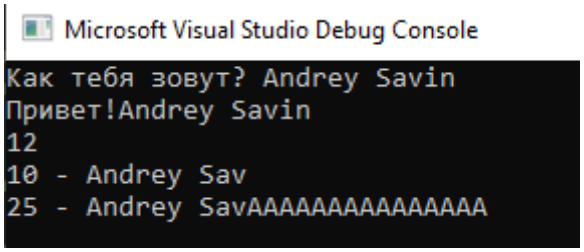

Рис. 27. Два варианта вызова метода resize()

Пример 24. Очистка строки clear(), проверка на пустоту empty().

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.clear();
cout << s.size() << " – " << s << endl;
if (s.empty())
        cout << "Строка пустая" << endl;
```

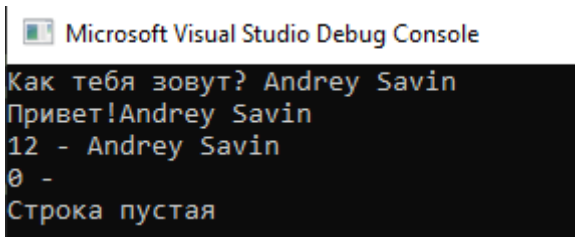

Рис. 28. Экран выполнения кода примера 24

Пример 25. Добавление символа в строку push_back().

```
s.push_back('S');
```

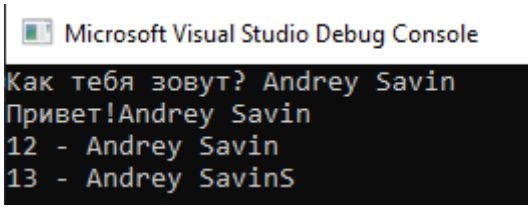

Рис. 29. Добавление символа в конец строки

Метод append() добавляет в конец строки несколько символов, другую строку или фрагмент другой строки. Имеет три способа вызова:

– s.append(n, c) – добавляет в конец строки n одинаковых символов, равных c. n имеет целочисленный тип, c – char;
– s.append(T) – добавляет в конец строки S содержимое строки T. T может быть объектом класса string или char[];
– s.append(T, pos, count) – добавляет в конец строки S символы строки T начиная с символа с индексом pos количеством count.

Пример 26. Вариант 1 – добавление n одинаковых символов в конец строки.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.append(10, 'S');
cout << s.size() << " – " << s << endl;
```

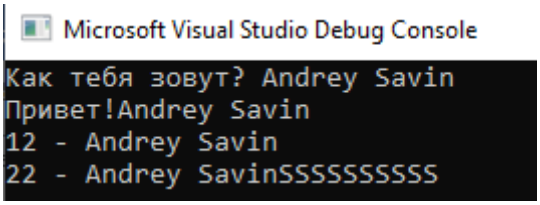

Рис. 30. Добавление n одинаковых символов в конец строки

Пример 27. Вариант 2 – добавление строки в конец исходной строки.

```
s.append(" Ivanovich");
```

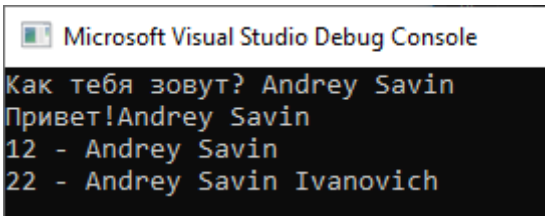

Рис. 31. Добавление строки в конец исходной строки

Пример 28. Вариант 3 – добавление строки в конец, начиная с символа на позиции 7 длиной 25 (в данном случае до конца строки).

```
s.append(" Mister Ivanovich", 7, 25);
```

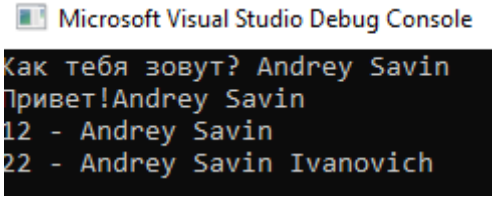

Рис. 32. Добавление части строки в конец исходной строки

Пример 29. Добавление строки в конец, начиная с символа на позиции 7 длиной 10.
s.append(" Mister Ivanovich Lozhkarev", 7, 10);

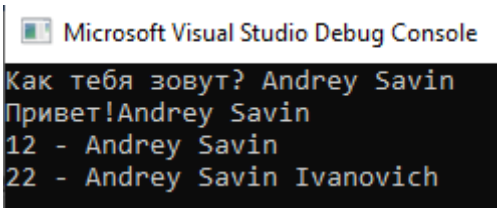

Рис. 33. Добавление части строки в конец исходной строки

Удаление символов из строки также возможно в нескольких вариантах:

– s.erase(pos) – удаляет из строки s с символа с индексом pos и до конца строки;
– s.erase(pos, count) – удаляет из строки s с символа с индексом pos количеством count или до конца строки, если pos + count > S.size().

Пример 30. Удаление символов, начиная с 12 позиции до конца.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.append(" Mister Ivanovich Lozhkarev", 7, 10);
cout << s.size() << " – " << s << endl;
s.erase(12);
cout << s.size() << " – " << s << endl;
```

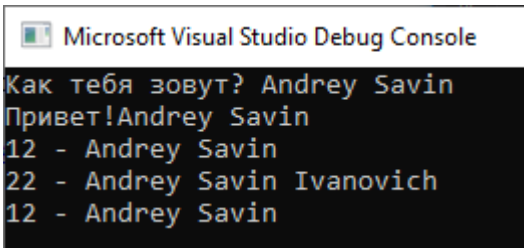

Рис. 34. Экран выполнения кода примера 30

Пример 31. Удаление 6 символов, начиная с 7 позиции.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.append(" Mister Ivanovich Lozhkarev", 7, 10);
cout << s.size() << " – " << s << endl;
s.erase(7, 6);
cout << s.size() << " – " << s << endl;
```

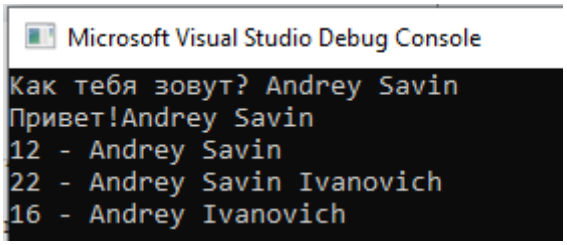

Рис. 35. Экран выполнения кода примера 31

Метод inset() вставляет в середину строки несколько символов, другую строку или фрагмент другой строки. Способы вызова аналогичны способам вызова метода append, только первым параметром является значение i – позиция, в которую вставляются символы. Первый вставленный символ будет иметь индекс i, а все символы, которые ранее имели индекс i и более сдвигаются вправо:

- s.insert(i, n, c) – вставить n одинаковых символов, равных c. n имеет целочисленный тип, c – char;
- s.insert(i, T) – вставить содержимое строки T. T может быть объектом класса string или char[];
- s.insert(i, T, pos, count) – вставить символы строки T начиная с символа с индексом pos количеством count.

Пример 32. Вставка 5 одинаковых символов S, начиная с 6 позиции.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.insert(6, 5, 'S');
cout << s.size() << " – " << s << endl;
```

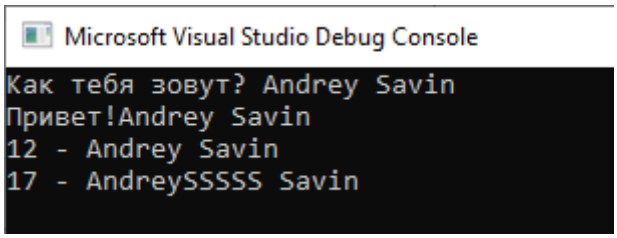

Рис. 36. Экран выполнения кода примера 32

Пример 33. Вставка подстроки, начиная с 6 позиции.

```
s.insert(6, " Ivanovich");
```

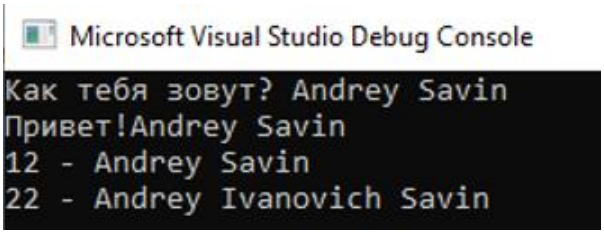

Рис. 37. Вставка подстроки

Пример 34. Вставка подстроки длиной 10, начиная с 6 позиции.

```
s.insert(6, " Ivanovich Lozhkarev", 10);
```

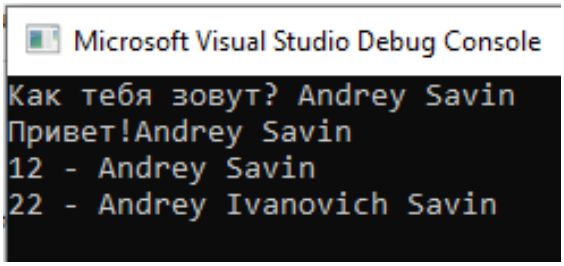

Рис. 38. Вставка подстроки

Добавление символов в строку типа string (методы insert, append, push_back) безопасно по сравнению с типом char[]. Длина строки при необходимости увеличится автоматически и выхода за границы массива не произойдет.

Выделение подстроки substr() позволяет скопировать в новую строку часть исходной строки:

- s.substr(pos) – возвращает подстроку данной строки начиная с символа с индексом pos и до конца строки;
- s.substr(pos, count) – возвращает подстроку данной строки начиная с символа с индексом pos количеством count или до конца строки, если pos + count > s.size().

Пример 35. Выделить подстроку, начиная с позиции 17 и до конца строки.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.insert(6, " Ivanovich Lozhkarev", 10);
cout << s.size() << " – " << s << endl;
string s1;
s1 = s.substr(17);
cout << s1.size() << " – " << s1 << endl;
```

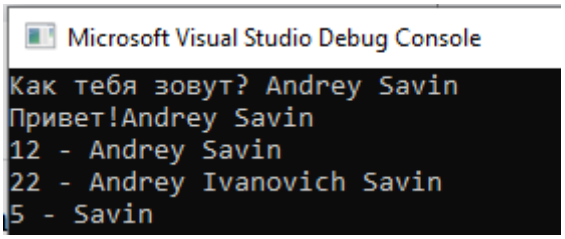

Рис. 39. Экран выполнения кода примера 35

Пример 36. Выделить подстроку длиной 10 символов, начиная с позиции 17.

```
s1 = s.substr(7, 10);
```

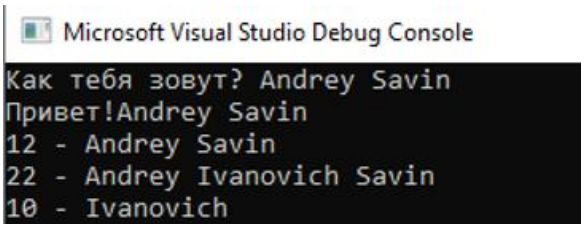

Рис. 40. Выделение подстроки

Метод замены символов replace() сочетает в себе удаление и вставку символов. Метод заменяет фрагмент строки на несколько равных символов, другую строку или фрагмент другой строки. Способы вызова аналогичны способам вызова метода append, только первыми двумя параметрами являются два числа: pos и count. Из данной строки удаляется count символов, начиная с символа pos, и на их место вставляются новые символы:

- s.replace(pos, count, n, c) – вставить n одинаковых символов, равных c. n имеет целочисленный тип, c – char;
- s.replace(pos, count, T) – вставить содержимое строки T. T может быть объектом класса string или char[];
- s.replace(pos, count, T, pos2, count2) – вставить символы строки T начиная с символа с индексом pos количеством count.

Пример 37. Начиная с 7 позиции заменить 9 символов исходной строки на 15 символов ‘S’.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.insert(6, " Ivanovich Lozhkarev", 10);
cout << s.size() << " – " << s << endl;
s.replace(7, 9, 15, 'S');
cout << s.size() << " – " << s << endl;
```

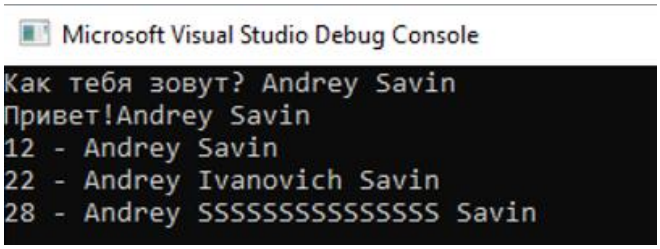

Рис. 41. Экран выполнения кода примера 37

Пример 38. Начиная с 7 позиции заменить 9 символов исходной строки на другую под-строку.

s.replace(7, 9, "Mihailovich");

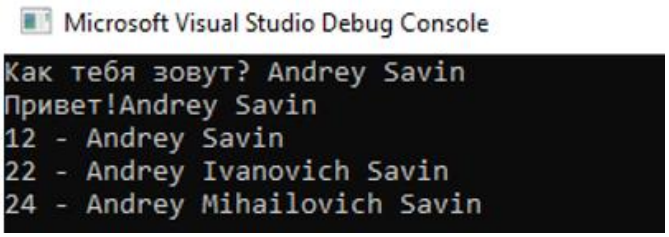

Рис. 42. Замена части строки

Пример 39. Начиная с 7 позиции заменить 9 символов исходной строки на часть другой подстроки из 11 символов, начиная с 0.

s.replace(7, 9, "Mihailovich Ложкарев", 0, 11);

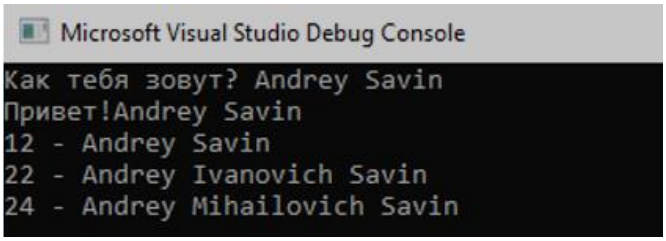

Рис. 43. Замена части строки на подстроку

Функция поиск подстроки find() ищет в данной строке первое вхождение другой строки str. Возвращается номер первого символа, начиная с которого далее идет подстрока, равная строке str. Если эта строка не найдена, то возвращается константа string::npos (которая равна -1, но при этом является беззнаковой, то есть на самом деле является большим безнаковым положительным числом).

Если задано значение pos, то поиск начинается с позиции pos, то есть возращаемое значение будет не меньше, чем pos. Если значение pos не указано, то считается, что оно равно 0 – поиск осуществляется с начала строки:

– s.find(str, pos = 0) – искать первое входение строки str начиная с позиции pos. Если pos не задано – то начиная с начала строки s;

– s.find(str, pos, n) – искать в данной строке подстроку, равную первым n символам строки str. Значение pos должно быть задано.

Пример 40. Найти подстроку, начиная с 0 символа.

```
string s;
cout << "Как тебя зовут? ";
getline(cin, s);
cout << "Привет!" << s << endl;
cout << s.size() << " – " << s << endl;
s.insert(6, " Ivanovich", 10);
cout << s.size() << " – " << s << endl;
int i = s.find("Ivan");
cout << i << endl;
```

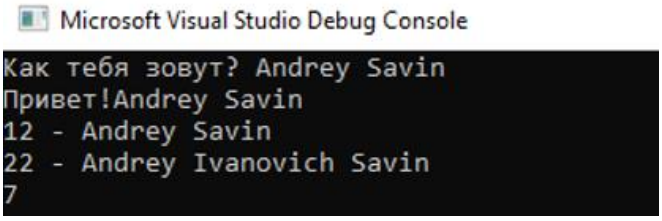

Рис. 44. Экран выполнения кода примера 40

Пример 41. Поиск подстроки.

```
int i = s.find("Mih");//подстрока не найдена
```

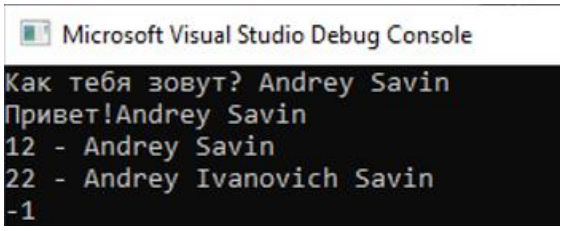

Рис. 45. Подстрока не найдена

Пример 42. Найти подстроку, начиная с 20 символа.

```
int i = s.find("Ivan", 20);
```

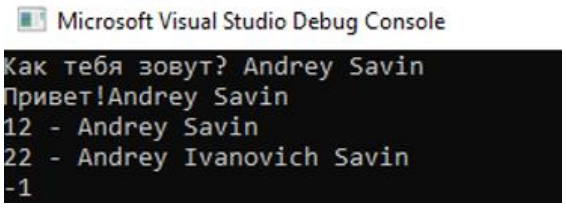

Рис. 46. Подстрока не найдена

Пример 43. Найти первые 4 символа подстроки, начиная с 0 символа.

```
int i = s.find("IvanPetr", 0, 4);
```

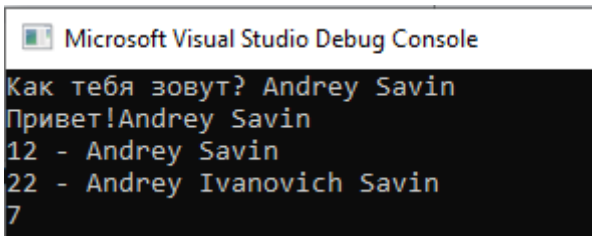

Рис. 47. Поиск подстроки

Аналогичная фунция rfind() – «правый» поиск.

Функции поиска символов подстроки find_first_of(), find_last_of() ищут в данной строке первое появление любого из символов данной строки str. Возвращается номер этого символа или значение string::npos (-1).

Если задано значение pos, то поиск начинается с позиции pos, то есть возращаемое значение будет не меньше, чем pos. Если значение pos не указано, то считается, что оно равно 0 – поиск осуществляется с начала строки.

S.find_first_of(str, pos = 0) – искать первое входение любого символа строки str начиная с позиции pos. Если pos не задано – то начиная с начала строки S.

Пример 44. Искать вхождение любого символа подстроки, начиная с 0 позиции.

```
int i = s.find_first_of("aIS");
cout << i << " – " << s[i] << endl;
```

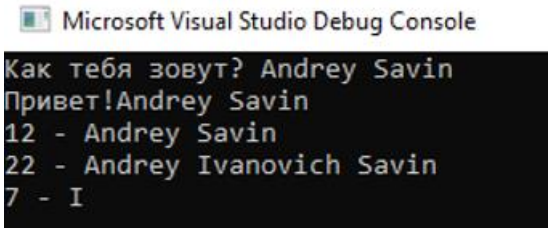

Рис. 48. Поиск любого символа подстроки слева

Пример 45. Искать вхождение любого символа подстроки, начиная с конца строки.

```
int i = s.find_last_of("aIS");
cout << i << " – " << s[i] << endl;
```

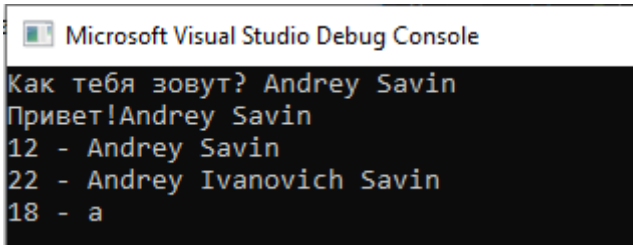

Рис. 49. Поиск любого символа подстроки справа

Строки типа string и строки типа char[] (так называемые С-строки) с точки зрения программы являются разными типами, хотя физически оба эти типа представляют собой наборы байтов с нуль-терминатором в конце. Для преобразования string в char[] существует функция c_str, которая возвращает указатель на область памяти, в которой хранятся символы строки (значение типа char*). Возвращаемое значение можно рассматривать как С-строку и использовать в функциях, которые должны получать на вход С-строку.

Пример 46.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
using namespace std;
int main() {
	setlocale(LC_ALL, "Russian");
	string s;
	cout << "Как тебя зовут? ";
	getline(cin, s);
	cout << "Привет!" << s << endl;
	cout << s.size() << " – " << s << endl;
	s.insert(6, " Ivanovich", 10);
	cout << s.size() << " – " << s << endl;
	char *s1 = new char [s.size() + 1];
	strcpy(s1, s.c_str());
	cout << s1 << endl;
}
```

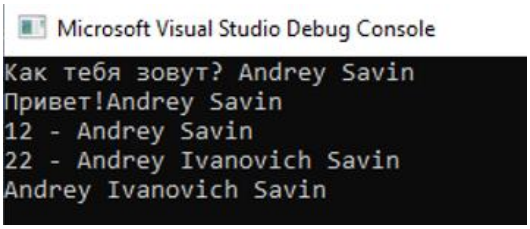

Рис. 50. Экран выполнения кода примера 46

4. ТЕКСТОВЫЕ ФАЙЛЫ

Файл – это именованная область данных.

Обычно файлы хранятся на внешних носителях, но также могут содержаться в оперативной памяти, являться по своей сути потоками или обеспечивать доступ к определённым устройствам. Способ организации и хранения файлов называется **файловой системой**.

Работа с файлами осуществляется средствами операционной системы. С точки зрения программиста файлы – это внешние ресурсы, предназначенные для ввода и вывода.

Работа с файлами всегда строится на трёх последовательных шагах:

- **открытие файла.** При этом операционная система предоставляет программе доступ к файлу, если это возможно, а также частично ограничивает доступ к этому файлу для других программ.
- **ввод из файла или вывод в файл.** При этом можно выяснить содержимое файла или наоборот записать в файл некоторую информацию.
- **закрытие файла.** При этом операционная система понимает, что файл больше не используется, и разблокирует доступ к нему для других программ.

Пример 47. Считывание и вывод строки посимвольно.

Сначала открывается текстовый файл input.txt, который был заранее создан в программе Блокнот и сохранен в папке проекта. Текст из файла считывается посимвольно, пока не будет достигнут конец файла. Затем полученная строка выводится на экран и сохраняется в другой текстовый файл output.txt.

```
#include <string>
#include <fstream>
#include <iostream>
using namespace std;
int main() {
	ifstream fin("input.txt");
	string s;
	if (fin.is_open()) {
		char c;
		while (fin.get(c)) {
			s = s + c;
		}
		fin.close();
	}
	cout << s << endl;
	ofstream fout("output.txt");
	if (fout.is_open()) {
		fout << s;
		fout.close();
	}
	return 0;
}
```

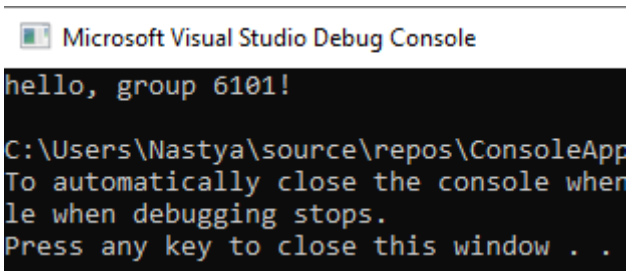

Рис. 51. Экран выполнения кода примера 47

В C++ для работы с файлами удобно использовать файловые потоки, объявленные в стандартном заголовочном файле fstream в пространстве имён std. Для ввода данных из файла используется поток класса ifstream, а для вывода данных в файл – поток класса ofstream**.**

Работа с этими потоками осуществляется точно так же, как и с консольными потоками cin и cout. Для ввода и вывода отдельных токенов используются соответственно операции >> и << для ввода и вывода отдельного символа можно использовать соответствовать методы get и put**.** Для ввода строки целиком до перевода строки используется функция getline**,** объявленная в заголовочном файле string. Для вывода перевода строки можно использовать объект endl, объявленный в заголовочном файле iostream.

Пример 48. Считывание и вывод строки типа string.

```
#include <string>
#include <fstream>
#include <iostream>
using namespace std;
int main() {
	string s1;
	ifstream fin("input.txt");
	getline(fin, s1);
	fin.close();
	cout << s1 << endl;
	ofstream fout("output.txt");
	if (fout.is_open()) {
		fout << s1;
		fout.close();
	}
	return 0;
}
```

Пример 49. Считывание и вывод строки типа char[].

```
#include <string>
#include <fstream>
#include <iostream>
using namespace std;
int main() {
	char s1[100];
```

```
        ifstream fin("input.txt");
        fin.getline(s1, 100);
        fin.close();
        cout << s1 << endl;
        ofstream fout("output.txt");
        if (fout.is_open()) {
                fout << s1;
                fout.close();
        }
        return 0;
}
```

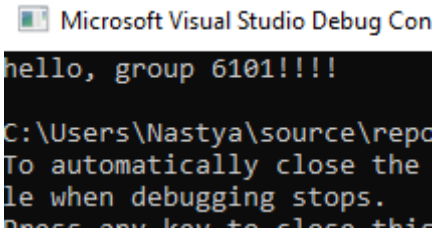

Рис. 52. Экран выполнения кода примеров 48 и 49

В примерах 48 и 49 читается содержимое файла input.txt в виде строки, после чего прочитанное записывается в файл output.txt. Названия файлов передаются в конструктор потоков при создании. В данном случае это относительные имена, поэтому будет произведена работа с файлами в текущей директории, но также можно использовать и абсолютные имена с указанием полного пути. После работы с файлом он закрывается путём вызова метода close.

При работе с файлами, как и с другими внешними ресурсами, нужно стараться как можно меньше времени держать их открытыми, ведь другие программы также могут нуждаться в работе с этими ресурсами. Типичная ошибка при работе с файлами – **отсутствие операции закрытия файла**, так что файл остаётся открытым даже когда это уже не требуется. При этом он остаётся заблокированным и работа с ним остаётся невозможной. Когда пользователь пытается изменить этот файл из другого приложения, операционная система сообщает ему об ошибке. Таких ситуаций следует избегать. Нужно как можно скорее после открытия файла выполнять все необходимые операции чтения или записи с ним, после чего сразу же его закрывать.

Пример 50. Считать из файла строку и подсчитать количество точек в ней. Строку и результат записать в выходной файл. Продублировать вывод на экран.

```
#include <string>
#include <fstream>
#include <iostream>
using namespace std;
int main() {
setlocale(LC_ALL, "Russian");
        char s[100];
```

```
        ifstream fin("input.txt");
        fin.getline(s, 100);
        fin.close();
        cout << s << endl;
        int i = 0, count = 0; //посчитаем количество точек в строке
        while (s[i] != '\0') {
            if (s[i] == '.') count++;
            i++;
        }
        ofstream fout("output.txt");
        if (fout.is_open()) {
            fout << s << endl;
            fout << "Исходная строка содержит " << count << " точек.";
            cout << "Исходная строка содержит " << count << " точек.";
            fout.close();
        }
        return 0;
}
```

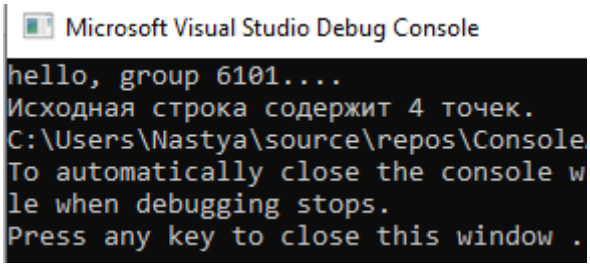

Рис. 53. Экран выполнения кода примера 50

Пример 51. Ввести одномерный числовой массив из текстового файла, отсортировать его и сохранить в текстовый файл и файл MS Excel.

```
#include <string>
#include <fstream>
#include <iostream>
using namespace std;
int main() {
    setlocale(LC_ALL, "Russian");
    double x;
    int n = 0;
    ifstream fin("input.txt");
    if (!fin.is_open()) { //проверка открытия файла
        cout << "Файл input не найден или не может быть открыт." << endl;
        return 1;
    }
    //подсчет чисел в файле
    while (fin >> x)
```

```
        n++;
cout << "Найдено чисел – " << n << endl;
fin.clear();
fin.seekg(0, ios::beg);//возврат на начало файла
double* a = new double[n];//создаем массив с нужным размером
if (NULL == a) {//проверка создания массива
        cout << "Ошибка создания массива." << endl;
        return 2;
}
cout << "Чтение массива из файла" << endl;
for (int i = 0; i < n; i++) {
        fin >> a[i]; //читаем элемент массива из файла
        cout << "a[" << i << "] = " << a[i] << endl;//выводим прочитанный элемент
        //на экран
}
fin.close();
//сортировка массива a методом выбора
for (int i = 0; i < n – 1; i++) {
        int imin = i;
        for (int j = i + 1; j < n; j++)
        if (a[j] < a[imin]) imin = j;
        double b = a[i];
        a[i] = a[imin];
        a[imin] = b;
}
//сохранение отсортированного массива в файл output.txt
ofstream fout("output.txt");
if (!fout.is_open()) { //проверка открытия файла
        cout << "Файл output не может быть открыт." << endl;
        return 1;
}
cout << "Отсортированный массив a" << endl;
fout << "Отсортированный массив a" << endl;
for (int i = 0; i < n; i++) {
        fout << i << " – " << a[i] << endl; //вывод в файл
        cout << "a[" << i << "] = " << a[i] << endl; //вывод на экран
}
fout.close();
ofstream ffout("test.csv");
if (!ffout.is_open()) { //проверка открытия файла
        cout << "Файл test не может быть открыт." << endl;
        return 1;
}
ffout << "Отсортированный массив a" << endl;
```

```
for (int i = 0; i < n; i++)
        ffout << i << ";" << a[i] << endl; //вывод в файл
ffout.close();
delete[]a;
return 0;}
```

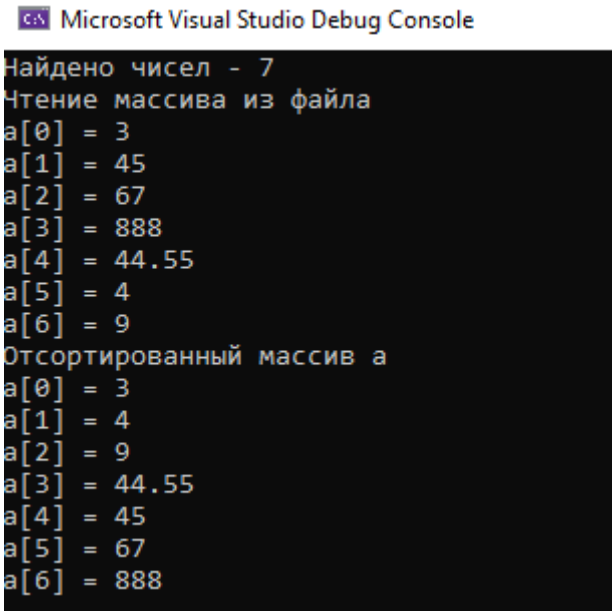

Рис. 54. Экран выполнения кода примера 51

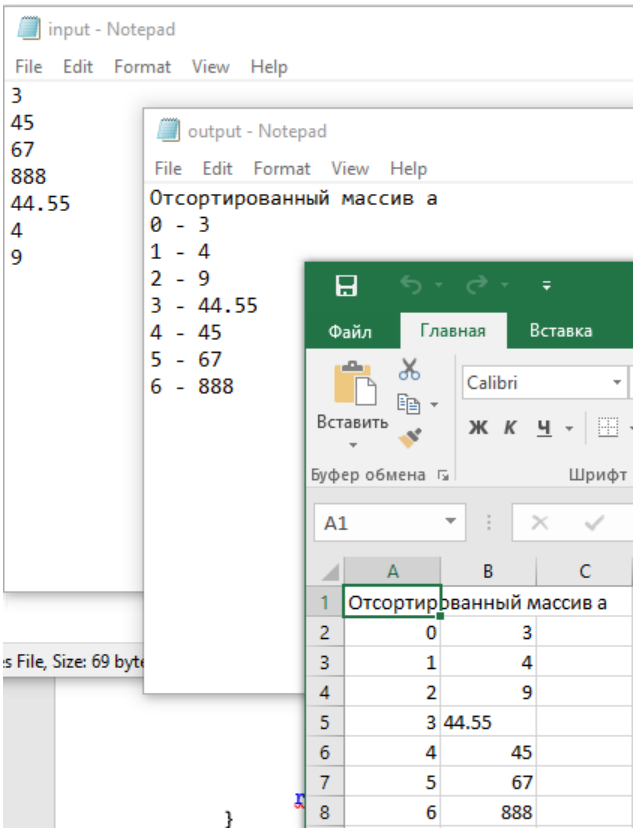

Рис. 55. Файлы, с которыми работает код примера 51

Пример 52. Считать строки из входного файла, записать их в выходной файл. Все считанные строки объединить в одну.

```
#define _CRT_SECURE_NO_WARNINGS
#include <string>
#include <fstream>
#include <iostream>
#include <Windows.h>
using namespace std;
const int N = 1024;//размер строки
int main() {
	//если в файле input.txt есть русские буквы, сохранить его в кодировке ANSI
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char s1[N], s_all[N] = "";
	int num = 0;
	ifstream fin("input.txt");
	ofstream fout("output.txt");
	if (!fin.is_open() || !fout.is_open()) { //проверка открытия и создания файлов
		cout << "File is not found or open." << endl;
		return 1;
	}
	while (fin.getline(s1, N)) {//будем читать из файла input.txt построчно
		cout << s1 << endl; //выводим строку на экран
		num++;
		strcat(s_all, s1); //присоединяем очередную строку в s_all
		fout << "Строка #" << num << ": " << s1 << endl;
		//записываем строку в output.txt с номером
	}
	fin.close();
	fout.close();
	cout << s_all << endl; //выводим на экран объединенную строку
	return 0;
}
```

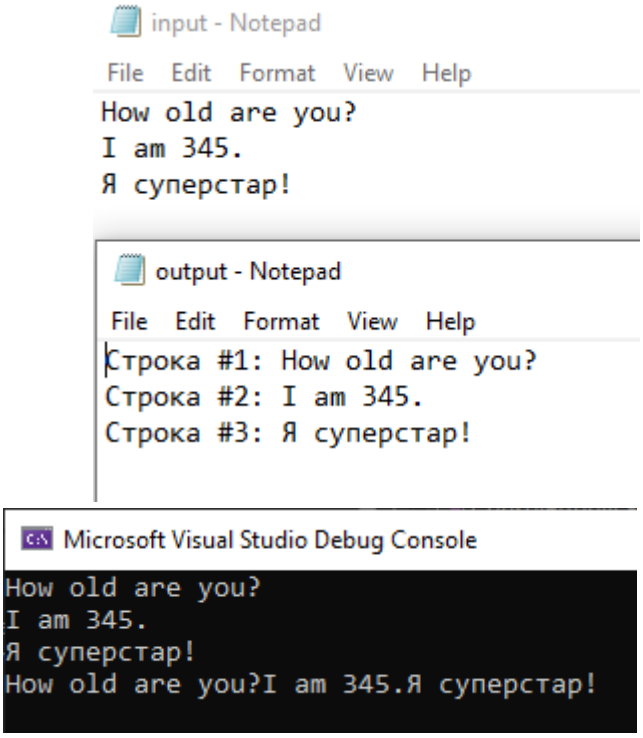

Рис. 56. Результаты работы кода примера 52

4.1. Лабораторная работа № 3. Текстовые файлы

Выполнить обработку текстовых файлов, содержащих строки ASCII, тип данных string, соблюдая следующие требования:

- исходный текст input.txt содержит несколько строк;
- выделить в текстовом файле все слова, разделенные символами-разделителями «_.,;:\n\t!?», и все слова, обладающие свойством Q, построчно записать в другой текстовый файл output.txt;
- за основу нахождения слов взять алгоритм с использованием массива flag (переработать его на тип string);
- индивидуальный алгоритм нахождения свойства Q, вывод заголовка, символьный пароль реализовать в виде функций.

Варианты заданий

1. Q: в слове нет арифметических знаков.
2. Q: в слове нет гласных латинских букв.
3. Q: слово состоит только из согласных латинских букв.
4. Q: в слове присутствуют и цифры, и арифметические знаки.
5. Q: в слове присутствуют и гласные, и согласные латинские буквы.
6. Q: в слове нет согласных латинских букв и цифр.
7. Q: в слове нет гласных латинских букв и арифметических знаков.
8. Q: в слове присутствуют гласные латинские буквы.
9. Q: в слове присутствуют согласные латинские буквы, но нет цифр.
10. Q: в слове присутствуют цифры, но нет арифметических знаков.

11. Q: в слове согласных латинских букв больше чем гласных.

12. Q: в слове количество цифр равно количеству арифметических знаков.

13. Q: в слове гласных латинских букв больше чем цифр.

14. Q: в слове количество гласных латинских букв равно количеству арифметических знаков.

15. Q: слово состоит только из гласных латинских букв и цифр.

16. Q: слово состоит только из согласных латинских букв и является регулярным.

17. Q: в слове нет согласных латинских букв, и оно является симметричным.

5. ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. Вывести на экран все цифры, строчные и прописные латинские буквы с их кодами.

2. Дано четное число n>0 и символы c1 и c2. Сформировать и вывести строку длиной n, которая состоит из чередующихся символов c1 и c2.

3. Вывести исходную строку наоборот.

4. Дана строка S размером n. Сформировать и вывести строку длиной 2n, которая состоит из символов строки S, между которыми вставлено по 1 пробелу.

5. Дана строка S. Определить, встречаются ли в ней 2 идущих подряд символа. Вывести такие символы и их позиции.

6. Подсчитать в строке S частоту вхождения каждого символа.

7. Дано целое положительное число. Перевести его в строку, не используя библиотечных функций.

8. Дана строка, изображающая двоичную запись целого положительного числа. Получить строку, изображающую десятичную запись этого числа.

9. В строке подсчитать количество слов, а в каждом слове его длину. Вывести на экран исходную строку и количество слов в ней. Затем вывести слова с их длинами по отдельности. Слова в строке разделяются пробелами.

10. Найти сумму имеющихся в исходной строке цифр. Затем удалить из строки все цифры.

11. Ввести две строки. Определить, можно ли составить вторую строку из символов первой.

12. Ввести строку, содержащую цифры. Заменить все цифры в ней соответствующими словесными написаниями.

13. Вывести на экран слова строки в обратном порядке.

14. Выделить все слова в строке скобками.

15. Выделить все слова в строке, где присутствуют цифры, скобками.

16. Вывести исходную строку с указанием перед каждым словом его порядкового номера в скобках.

17. Увеличить расстояние между словами исходной строки на 2 пробела.

СПИСОК ЛИТЕРАТУРЫ

1. Иванова, Г.С. Основы программирования: учебник для вузов / Г.С. Иванова. – Москва: Изд-во МГТУ им. Н.Э. Баумана, 2016. – 416 с.

2. Павловская, Т.А. C/C++. Программирование на языке высокого уровня / Т.А. Павловская. – Санкт-Петербург: Питер, 2021. – 461 с.

3. Рацеев, С.М. Программирование на языке Си: учебное пособие для вузов / С.М. Рацеев. – Санкт-Петербург: Лань, 2022. – 332 с.

4. Рацеев, С.М. Программирование. Лабораторный практикум: учебное пособие для вузов / С.М. Рацеев. – Санкт-Петербург: Лань, 2023. – 104 с.

5. Страуструп, Б. Язык программирования C++ / Б. Страуструп. – Санкт-Петербург: Москва: «Невский Диалект» – «Издательство БИНОМ», 1999. – 991 с.