Символьные массивы в языке C++. Работа со строками

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Цели и задачи лекции

Цель лекции – изучить использование массивов в языке программирования C++

Теоретическая часть

В языке программирования С заложены средства для задания последовательностей упорядоченных данных. Такие последовательности называются массивами. В массивах должны быть упорядочены данные одного и того же типа. В данной лабораторной работе будут рассматриваться массивы символов, которые определятся типом char. Одномерный массив наиболее часто применяется в виде строки символов.

Теоретическая часть

Строка – это одномерный массив символов, заканчивающийся нулевым символом [1; 2]. В языке С признаком окончания строки служит символ '\0'. При объявлении массива символов, предназначенного для хранения строки, необходимо отвести одно место для нуля, т.е. для символа окончания строки '\0'.

Теоретическая часть

Например, если дана строка qwerty, в которой 6 символов, каждый из которых занимает в памяти 1 байт, то при инициализации такой строки необходимо отвести 1 байт для нулевого символа. Поэтому следует сделать объявление строки для семи символов:

```
char str[7] = "qwerty";
```

Теоретическая часть

Альтернативным объявлением может служить безразмерная инициализация:

```
Char str[ ] = "qwerty";
```

При этом в случае определения длины строки результатом будет число 6. Размер строки не изменится, если в ней указать символ окончания строки:

```
char str[ ] = "qwerty\0";
```

Теоретическая часть

Аналогично числовым массивам в языке С могут использоваться массивы строк, т.е. набор одномерных массивов символов. Например, сервер базы данных сверяет команды пользователей с массивом допустимых команд. В качестве массива строк для этого случая будет служить двухмерный символьный массив. Размер левого измерения определяет количество строк, а правого – максимальную длину каждой строки.

Например:

```
char str[30][80];
```

Число 30 – это количество строк массива, а число 80 – максимальная длина каждой строки с учетом нулевого символа завершения строки.

Чтобы обратиться к отдельной строке двухмерного символьного массива, достаточно указать только левый индекс объявленного массива.

Теоретическая часть

Многомерные символьные массивы образуются, как и числовые массивы:

```
char str[n][m]...[N];
```

В объявлении массива n – первая размерность, m – вторая размерность, ..., N – N -я (последняя) размерность. Значения размерностей – целые неотрицательные числа.

Одномерные символьные массивы – строки

Одномерный массив – это список связанных однотипных переменных.

Общая форма записи одномерного массива:

тип имя_массива[размер];

В приведенной записи элемент тип объявляет базовый тип массива. Количество элементов, которые будут храниться в массиве с именем имя_массива, определяется элементом размер.

Одномерные символьные массивы – строки

В языке С индексация массива (символьного или числового) начинается с нуля.

Доступ к отдельному элементу массива осуществляется с помощью индекса. Индекс описывает позицию элемента внутри массива.

Все массивы занимают смежные ячейки памяти, т.е. элементы массива в памяти расположены последовательно друг за другом.

Одномерные символьные массивы – строки

Ячейка памяти с наименьшим адресом относится к первому элементу массива, а с наибольшим – к последнему.

Для массива символов при инициализации массива необходимо резервировать место для символа окончания строки, т.е. для символа '\0'.

Строковая константа – это набор символов, заключенных в двойные апострофы, например, "hello".

Одномерные символьные массивы – строки

В конец символьной строки не требуется обязательно добавлять нуль, компилятор языка С делает это автоматически. При инициализации символьной строки как одномерного массива необходимо предусмотреть место для нулевого символа, например:

```
char str[7] =  "hello";
```

Одномерные символьные массивы – строки

Каждая строка содержит на один символ больше, чем задано явно: все строки оканчиваются нулевым символом, имеющим значение 0.

Для одномерных массивов общий размер массива в байтах вычисляется по формуле:

всего байт = размер типа в байтах * количество элементов

Двухмерные символьные массивы

Двухмерный массив представляет собой список одномерных массивов.

Общая форма записи двухмерного массива:

тип имя_массива[размер1] [размер2];

В приведенной записи размер1 означает число строк двухмерного массива, а размер2 – количество столбцов.

Двухмерные символьные массивы

При этом размерность размер2 определяет максимальную длину для заданного массива. Обычно размер2 задают с некоторым запасом.

В двухмерном массиве позиция любого элемента определяется двумя индексами. Индексы каждого из размеров массива начинаются с нуля.

Двухмерные символьные массивы

Место хранения для всех элементов массива определяется во время компиляции. Память, выделенная для хранения массива, используется в течение всего времени существования массива.

Для двухмерных массивов заданного типа общий размер массива в байтах вычисляется по формуле:

всего байт = количество строк * количество

Двухмерные символьные массивы

Инициализация двухмерного символьного массива может быть определена либо посимвольно, либо построчно, например

```
char str[3][80] = {
        {'1','2','3','4','5'},
        {'x','y','z'},
        {'A','B','C','D'}
          };
```

Двухмерные символьные массивы

```
char str2[3][80] = {
        "0123456789",
        "x_y_z",
        "A B C D" };
```

Число 80 взято с запасом для возможной длины строки. Число 3 – это количество строк двухмерного массива.

Двухмерные символьные массивы

В обоих случаях могут быть добавлены символы окончания строки ('\0'). Символ '\0' не выводится на экран дисплея и не передается в файл, например, в текстовый файл. В то же время необходимо помнить, что каждая строка заканчивается нулевым символом.

Многомерные символьные массивы

Общая форма записи многомерного массива:

тип имя_массива[размер1] [размер2]... [размерN];

Индексация каждого размера начинается с нуля. Элементы многомерного массива располагаются в памяти в порядке возрастания самого правого индекса. Поэтому правый индекс будет изменяться быстрее, чем левый (левые). При этом в конце каждой строки

Многомерные символьные массивы

При обращении к многомерным массивам компьютер много времени затрачивает на вычисление адреса, так как при этом приходится учитывать значение каждого индекса. Поэтому доступ к элементам многомерного массива происходит значительно медленнее, чем к элементам одномерного. В связи с этим использование многомерных массивов встречается значительно реже, чем одномерных или двухмерных массивов.

Многомерные символьные массивы

Для многомерных массивов общий размер многомерного массива в байтах вычисляется по формуле:

всего байт = размер1 * размер2* … *размерN * размер типа в байтах

Очевидно, многомерные массивы способны занять большой объем памяти, а программа, которая их использует, может очень быстро столкнуться с проблемой нехватки памяти.

Многомерные символьные массивы

Для определения размера типа в байтах применяется функция **sizeof()**, которая возвращает целое число. Например, **sizeof(char)**. При инициализации многомерных массивов необходимо указать все данные (размерности) за исключением крайней слева размерности. Это нужно для того, чтобы компилятор смог определить длину подмассивов, составляющих массив, и смог выделить необходимую память.

Пример 1. Напишите программу определения длины заданных строк и их распечатки, а также определения размера строк в байтах.
Для решения поставленной задачи применим библиотечную функцию **strlen()** и оператор **sizeof**.
Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
```

Практическая часть

```
int main (void) {
    char str[] = {'A','B','C','D','\0'};
    char str2[] = "hello, world\0";
    printf("\n\t The lines are:\n\n\t ");
puts(str);
printf("\t "); // Для отступа от края дисплея
puts(str2);
```

Практическая часть

```
printf("\n\t The length of the 1st line (%s) is: %i\n", str, strlen(str));
printf("\t The size of the memory of the 1st line (%s) is: %i\n", str, sizeof str);
printf("\n\t The length of 2-nd line (%s) is: %i\n", str2, strlen(str2));
printf("\t Memory size 2-nd line (%s) is: %i\n", str2, sizeof str2);
```

Практическая часть

```
    printf("\n Press any key: ");
    _getch();
    return 0;
}
```

В программе функция strlen() возвращает длину строки, причем строка должна заканчиваться символом строки. Символ конца строки не учитывается.

Практическая часть

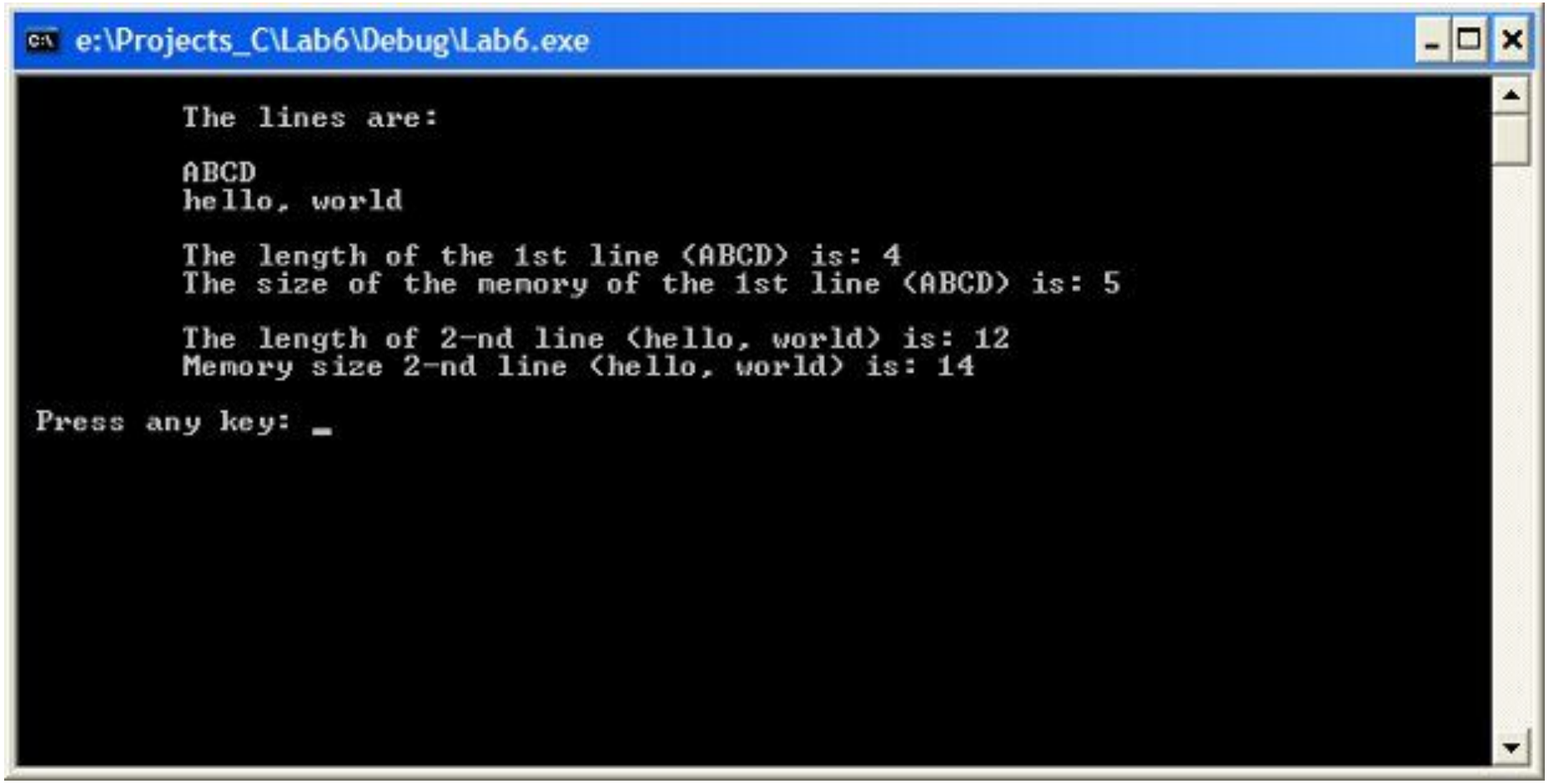
Для работы с функцией **strlen()** необходимо подключить заголовок **<string.h>.** Оператор **sizeof** во время компиляции программы получает размер типа или значения. Для определения размера типа оператор используется со скобками, например, **sizeof(char),** а для определения размера конкретного значения оператор может использоваться без скобок.

Практическая часть

В программе использована функция **puts()**, которая записывает строку, адресуемую, например, параметром **str**, в стандартное выходное устройство – дисплей. Символ конца строки преобразуется в разделитель строк.

Результат выполнения программы показан на рисунке.

Практическая часть



Пример 2. Напишите программу копирования одной заданной строки в другую.

Для решения используем библиотечную функцию **strcpy(),** для которой подключим заголовок **<string.h>.**

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

int main (void) {
   char str1[] = "123456789";
   char str2[] = "qwerty";
```

Практическая часть

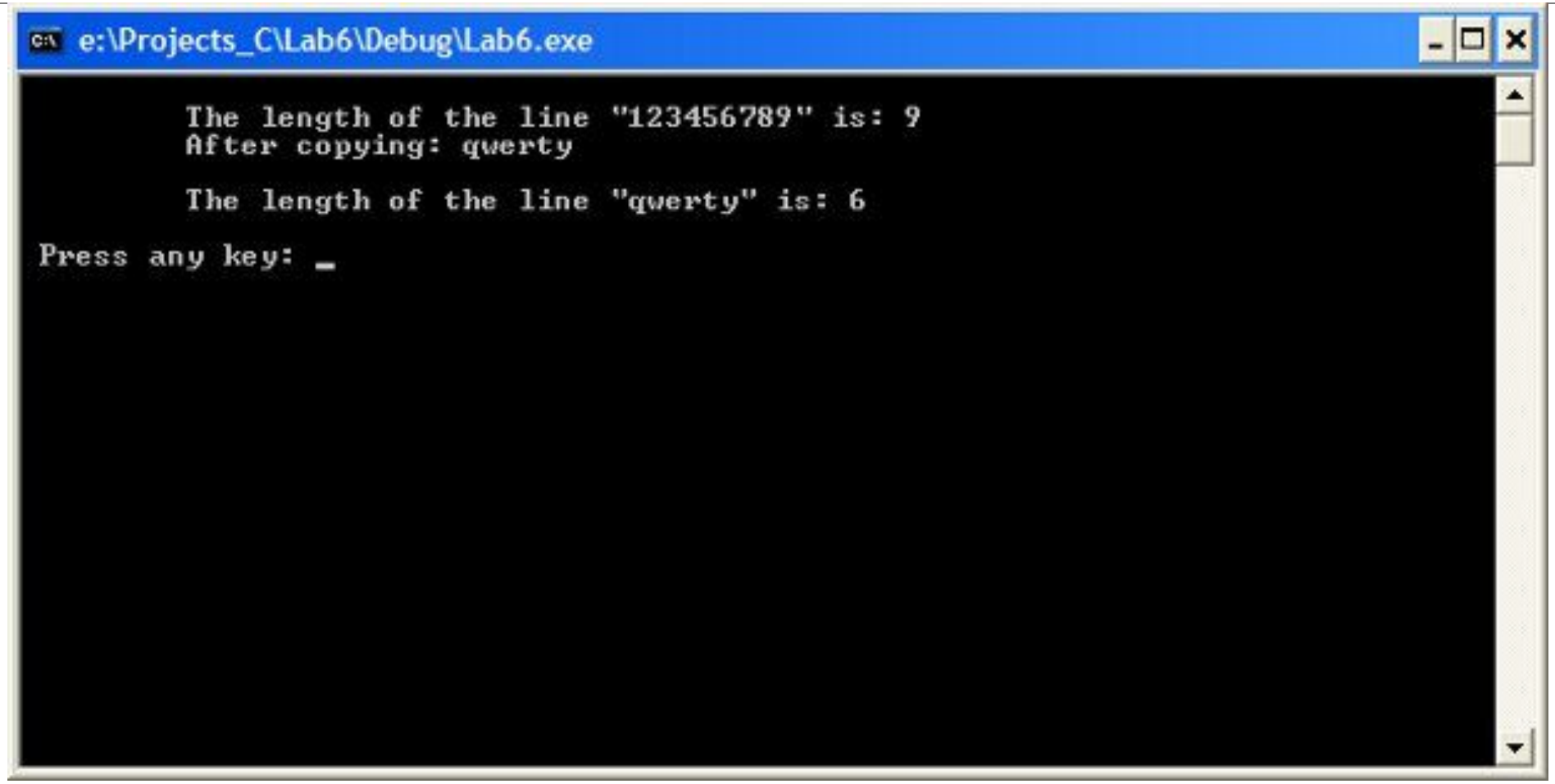
```
printf("\n\t The length of the line \"%s\" is: %d\n", str1,
strlen(str1));
        strcpy(str1, str2);
    printf("\t After copying: ");
    puts(str1);
printf("\n\t The length of the line \"%s\" is: %d\n", str1,
strlen(str1));
```

Практическая часть

```
    printf("\n Press any key: ");
    _getch();
    return 0;
}
```

Результат выполнения программы показан на рисунке.

Практическая часть



Пример 3. Напишите программу преобразования десятичной системы счисления в двоичную. Исходное десятичное число считайте целым без знака.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
```

Практическая часть

```
int main (void) {
    const char D[] = {
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
        int newNumber[80];
        long int inputNumber;
        int i = 0, base = 2;
    printf("\n\t Enter a positive integer: ");
        scanf_s("%ld", &inputNumber);
```

Практическая часть

```
// Прямой процесс преобразования десятичного числа в двоичное
        do {
            newNumber[i] = inputNumber % base;
            ++i;
            inputNumber /= base;
        } while ( inputNumber != 0 );
```

Практическая часть

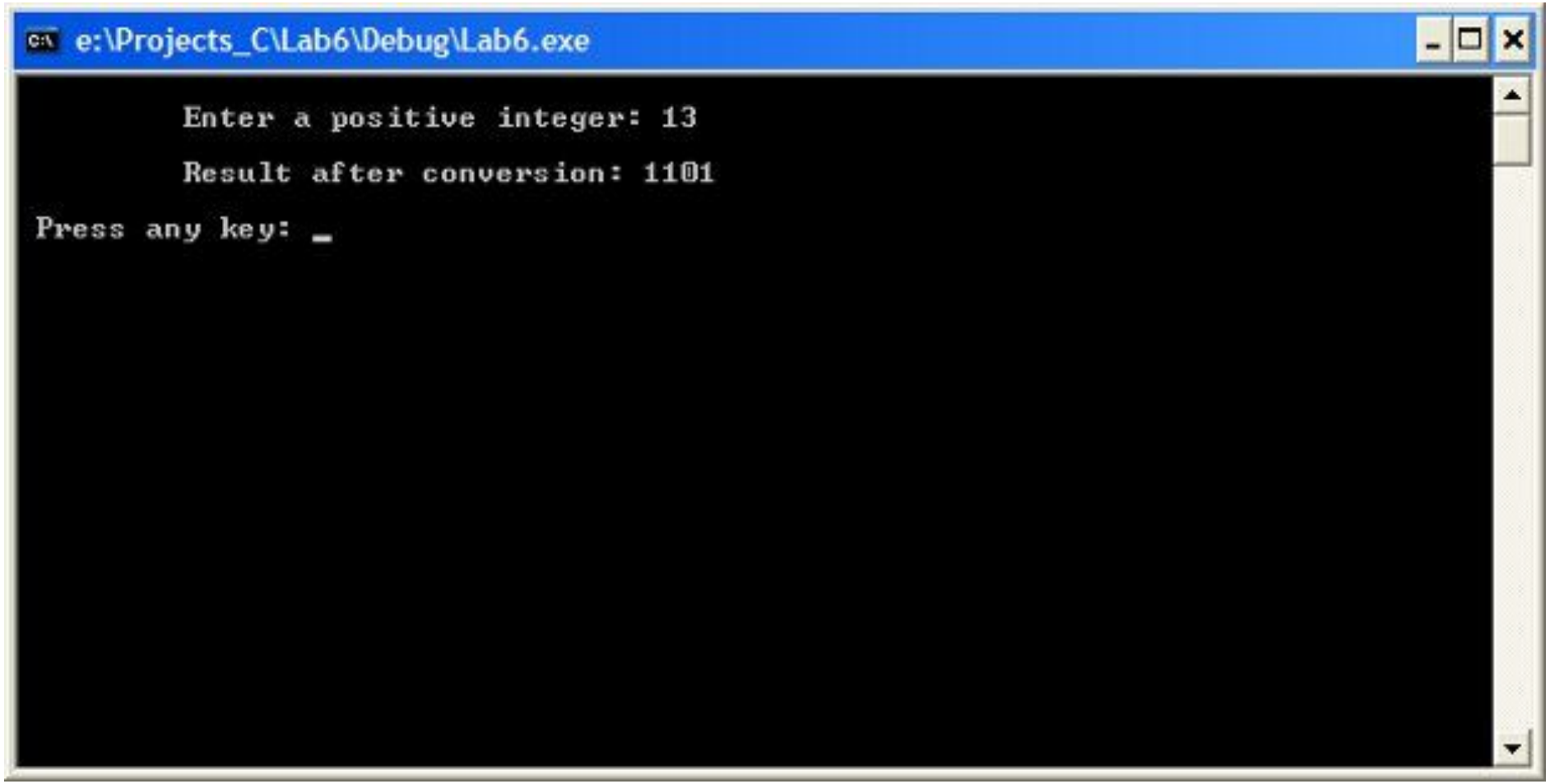
```
//Запись преобразованного числа в обратном порядке
        printf("\n\t Result after conversion: ");
for (--i; i >= 0; --i)
        printf("%d", newNumber[i]);
        printf("\n\n Press any key: ");
    _getch();
    return 0;
}
```

Практическая часть

В программе использован квалификатор (спецификатор) типа **const**, который указывает компилятору, что символьный массив не может изменяться в программе.

Результат выполнения программы показан на рисунке.

Практическая часть



Пример 4. С помощью функции **strcat()** присоедините одну строку к другой с пробелом и без.

Пусть имеются строки str1 и str2. Функция **strcat()** присоединяет к строке str1 копию строки str2 и завершает строку str1 нулевым символом. Процесс присоединения называется конкатенацией.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define N 79

int main (void) {
   char str1[N+1], str2[N+1];
```

Практическая часть

```
printf("\n\t Print 1 string of characters: ");
 gets_s(str1, N); // для MS Visual Studio

printf("\t Print 2 string of characters: ");    gets_s(str2, N); // для MS Visual Studio

strcat_s(str1, N, str2); // для MS Visual Studio
printf("\n\t Result after concatenation: %s\n", str1);
```

Практическая часть

```
    printf("\n Press any key: ");
   _getch();
   return 0; }
```

Возможный результат выполнения программы показан на рисунке.

Практическая часть

Спасибо за внимание