|  |
| --- |
| **ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ** |
|  |
|  |
|  |
| **Практическое занятие №3.**  **Тема: Ввод-вывод и форматирование данных** |
|  |
|  |
|  |

**Цель:** получение практических навыков работы с файлами и использования средств форматирования данных.

**Основные теоретические положения**

**Типы данных. Строки**

**Определение и ввод-вывод строк**

В отличие от других языков программирования, в классическом языке Си отсутствует строковый тип данных. Вместо него строка символов представляется в памяти компьютера как массив элементов типа **char**, в конце которого помещен нуль-символ**’\0’**. Таким образом, строка – это последовательность символов, оканчивающаяся нуль-символом. Нуль-символ как шестнадцатеричная константа представляется кодом**0x00**.

Для задания строки массив символов при объявлении инициализируется строковой константой. Это можно сделать по-разному. Следующие три строки эквивалентны:

**char str[ ] = ”Слово”;**

**char str[6] = ”Слово”;**

**char str[6] = {’С’, ’л’, ’о’, ’в’, ’о’,’\0’};**

Размер массива должен быть на единицу больше числа символов в строке.

Для того чтобы вывести символьную строку на экран, достаточно в операторе вывода указать только имя символьного массива:

**cout<<str;**

Ввод символьной строки осуществляется аналогично: в операторе ввода следует указать имя символьного массива:

**cin>>str;**

Пример, демонстрирующий ввода-вывода символьной строки:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**char st[30];**

**cout<<“Insert information of symbols: “<<endl;**

**cin>>st;** // ввод с клавиатуры

**cout<<st;** // вывод на экран

**return 0;**

**}**

В данном примере объявляется 30-элементный символьный массив. Чтобы его заполнить применяется простая инструкция – **cin>>st;**затем строка выводится на экран. Следует отметить, что если при вводе символов ввести пробел, то после пробела данные в программу поступать не будут, т.к. ввод осуществляется до первого разделителя, которым является пробел.

Для устранения данного неудобства применяется функция ввода строк

**gets(char \*string),**

где **string** задает область памяти, в которую помещаются символы вводимой строки. Для вывода на экран используется симметричная функция

**puts(char \*string)**.

Пример использования указанных функций:

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**char str [20];**

**cout<< (“Insert information: ”);**

**gets (str);**

**puts (str);**

**return 0;**

**}**

Строки можно формировать в массивы. Массив строк — это специальная форма двумерного массива символов. Например,

**char str\_array[3][20];**

Данный массив предназначен для хранения 3 строк длиной 20 символов.

Задать массив символьных строк можно следующим способом:

**char str\_array[3][20]={"Пример","задания","строк"};**

Получить доступ к отдельной строке довольно просто: достаточно указать только левый индекс. Например, вывод третьей строки массива **str\_array** производится следующим образом:

**cin>>str\_array[2];**

Пример, в котором формируются некоторые анкетные данные для трёх человек:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**char name[3][10];**

**char profession[3][25];**

**int age[3];**

**for(int i=0; i<3; i++)**

**{**

**cout<<"Введите фамилию, профессию, возраст: ";**

**cin>> name[i];**

**cin>> profession[i];**

**cin>>age[i];**

**}**

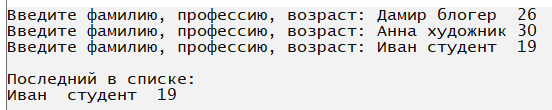
**cout<<"\n Последний в списке: \n";**

**cout<<name[2]<<" "<<profession[2]<<" "<<age[2]<<'\n';**

**return 0;**

**}**

Вид экрана:



**Динамическое распределение памяти**

Система динамического распределения памяти — это средство получения программой некоторой области памяти во время ее выполнения.

Язык C++ содержит два оператора, **new** и **delete**, которые выполняют функции по выделению и освобождению памяти. Их формат:

*переменная-указатель* = **new** *тип\_переменной*;

**delete** *переменная-указатель*;

Оператор **new** позволяет динамически выделить область памяти.

Оператор **delete** освобождает ранее динамически выделенную память.

Пример программы, которая иллюстрирует использование операторов **new** и **delete**:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int \*p;**

**p = new int;** // Выделение памяти для int-значения

**\*p = 20;** // Занесение в эту область памяти значения 20

**cout<< \*р;**

**delete р;** // Освобождение памяти.

**return 0;**

**}**

Для выделения памяти под массив следует указать его размерность. В следующем примере приведена функция, в которой выделяется память под символьный массив, в который затем копируются значения. В конце производится освобождение выделенной области памяти.

**void employee (int num, char\* st)**

**{**

**int number=num;**

**char\*s=new char[num];**

**for(int i=0; i<num; i++)**

**s[i]=st[i];**

**cout<<s<<' '<<number<<**'\n'**;**

**delete []s;**

**}**

**Передача символьных строк между функциями**

Чтобы передать символьную строку в функцию, следует в качестве параметра функции указать символьный массив, а при вызове функции использовать имя передаваемой символьной строки.

Пример.

|  |  |
| --- | --- |
|  | **#include <iostream>**  **using namespace std;**  **void get\_array(char s[])**  **{**  **s[0]='b';** // Изменили первый символ  **cout<<s<<**'\n'**;**  **}**  **int main()**  **{**  **char str[]="kasta";** // Инициализируем строку  **get\_array(str);** // Передали массив в функцию  **return 0;**  **}** |

**Операции со строками**

Помимо ввода/вывода со строками с помощью функций можно производить и другие операции. Функции, проводящие операции со строками, определены в заголовочном файле <cstring>.

Наиболее часто используются следующие четыре функции.

**strcpy(str1,str2)** – копирует строку str2 в строку str1, включая ’\0’; возвращает str1. *Здесь и далее str1 и str2 принадлежат типу char\*.*

**strcat(str1,str2)** – приписывает str2 к str1; возвращает str1.

**strcmp(str1,str2)** – сравнивает str1 и str2; возвращает <0, если str1<str2; 0, если str1= str2; и >0, если str1>str2 (сравнение здесь - лексикографическое).

**strlen(str)** – возвращает длину str.

**Strncat(str1, str2, n)** – приписывает первые n элементов к str1.

Пример использования данных функций:

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**int main()**

**{**

**char str1[50];**

**char str2[36];**

**strcpy(str1,"To be ");**

**strcpy(str2,str1);**

**strcat(str1, "continue ...");**

**char \*ps= "Fairground";**

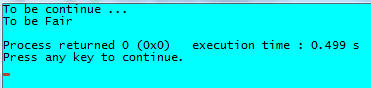
**strncat(str2, ps, 4);**

**cout<< str1 <<**'\n'**<<str2<<**'\n'**;**

**return 0;**

**}**

На экран будет выведено:



Функции, проводящие операции со строками, приведены в таблице 3.

Таблица 3

| **№** | **Функция** | **Описание** |
| --- | --- | --- |
| 1. | strcat(str1,str2) | Присоединяет str2 к str1, возвращает str1 |
| 2. | strncat(str1,str2,n) | Присоединяет не более n символов str2 к str1, завершает строку символом '\0', возвращает str1 |
| 3. | strсpy(str1,str2) | Копирует строку str2 в строку str1, включая '\0', возвращает str1 |
| 4. | strncpy(str1,str2,n) | Копирует не более n символов строки str2 в строку str1, возвращает str1 |
| 5. | strcmp(str1,str2) | Сравниваетstr1и str2, возвращает значение 0, если строки эквивалентны |
| 6. | strncmp(str1,str2,n) | Сравнивает не более n символов строк str1 и str2, возвращает значение 0, если начальные n символов строк эквивалентны |
| 7. | strlen(str) | Возвращает количество символов в строке str |
| 8. | strset(str,c) | Заполняет строку str символами, код которых равен значению c, возвращает указатель на строку str |
| 9. | strnset(s,c,n) | Заменяет первые n символов строки str символами, код которых равен c, возвращает указатель на строку str |
| 10. | strchr(str,symb) | Ищет символ symb в строке str и возвращает указатель на первое совпадение |
| 11. | strrchr(str,symb) | Ищет символ symb в строке str и возвращает указатель на последнее совпадение |
| 12. | strcspn(str1,str2) | Выполняет поиск первого вхождения в строку str1 любого из символов строки str2 и возвращает количество символов до найденного первого вхождения |
| 13. | strpbrk(str1,str2) | Выполняет поиск первого вхождения в строку str1 любого из символов строки str2 и возвращает указатель на найденный символ |
| 14. | strspn(str1,str2) | Поиск символов строки str2 в строке str1. Возвращает длину начального участка строки str1, который состоит только из символов строки str2 |
| 15. | strstr(str1,str2) | Ищет первое вхождение подстроки str2 в строке str1 |
| 16. | strtok(str,delim) | Ищет символы-разделители delim в строке str, возвращает указатель на последний найденный |
| 17. | memchr(str,c,n) | Ищет первое вхождение символа c в первых n байтахstr |
| 18. | memcpy(dest,src,n) | Копирует n символов из src в dest |
| 19. | memset(str,c,n) | Копирует символ cв первые n символов строки str |
| 20. | memcmp(str1,str2,n) | Сравнивает первые n байтов str1 и str2 |

В библиотеке **<cstring>** присутствуют функции поиска. Так, функция **strchr(s,chr)** производит поиск первого вхождения символа **chr** в строке **s**. В случае удачного поиска функция возвращает указатель на место первого вхождения символа **chr**. Если символ не найден, то возвращается ноль.

Рассмотрим пример использования функции **strchr(s, chr)**, в котором в строку "Seven Zero Null" необходимо перед словом "Null" вставить слово "One".

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**int main()**

**{**

**char cs[20] = "Seven Zero Null";**

**char str[5];**

// определяем место первого вхождения (указатель) буквы 'N' в строке

**char\*sn=strchr(cs,'N');**

**strncpy(str,sn,4);** // копируем "Null" в str

**strcpy(sn,"One ");** // на место "Null" ставим "One "

**strcat(cs,str);** // добавляем в конец строки "Null"

**cout<<cs<<endl;**

**return 0;**

**}**

|  |  |
| --- | --- |
| Вид экрана: |  |

**Тип данных «строка»**

В современном стандарте C++ определен класс с функциями для организации работы со строками, который подключается с помощью директивы:

#include<string>.

В нем содержится ряд функций для работы со строками. Например, **size()** определяет длину строки, а **insert()** позволяет делать вставки в строку.

Строковая переменная, например st, объявляется следующим образом:

**string st;**

Некоторые возможности, которыми обладает класс string.

***Инициализация строк*** при описании и ***длина строки*** (не включая завершающий нуль-символ):

**string st( "Моя строка\n" );**

**cout<< "Длина " <<st<< ": " <<st.size()**

**<< " символов, включая символ новой строки\n";**

Строка может быть задана и пустой:

**string st2;**

Для проверки того, ***пуста ли строка***, можно сравнить ее длину с 0:

**if (!st.size())** // пустая

или применить метод **empty(),** возвращающий **true** для пустой строки и **false** для непустой:

**if (st.empty())** // пустая

Третья форма создания строки инициализирует объект типа **string** другим объектом того же типа:

**string st3( st );**

Для ***сцепления строк*** используется операция сложения (+) или операция сложения с присваиванием (+=). Пусть даны две строки:

**string s1("Привет, ");**

**string s2("Санкт-Петербург\n");**

Мы можем получить третью строку, состоящую из конкатенации первых двух, таким образом:

**string s3 = s1 + s2;**

Если же мы хотим добавить s2 в конец s1, мы должны написать:

**s1 += s2;**

При обработке строк типа string можно использовать следующие функции:

* **s.length()** — возвращает длину строки s;
* **s.substr(pos, length)** — возвращает подстроку из строки s, начиная с номера pos длиной length символов;
* **s.empty()** — возвращает значение true, если строка s пуста, false — в противном случае;
* **s.insert(pos, s1)** — вставляет строку s1 в строку s, начиная с позиции pos;
* **s.remove(pos, length)** — удаляет из строки s подстроку length длиной pos символов;
* **s.find(s1, pos)** — возвращает номер первого вхождения строки s1 в строку s, поиск начинается с номера pos, параметр pos может отсутствовать, в этом случае поиск идёт с начала строки;
* **s.findfirst(s1, pos)** — возвращает номер первого вхождения любого символа из строки s1 в строку s, поиск начинается с номера pos, параметр pos может отсутствовать, в этом случае поиск идёт с начала строки.

Ниже приводится пример программы, работающей со **string**.

**include<iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**string s = "Test";**

**s.insert (1,"r");**

**cout<< s <<endl;**

**string \*s2 = new string("Hugo");**

**cout<<s2->size();**

**return 0;**

**}**

Строчка программы **cout<<s2->size()** демонстрирует синтаксис, когда используется указатель на строку. Она равносильна записи:

**cout<< (\*s2).size().**

**Типы данных. Структуры**

**Общие сведения о структурах**

Структура – это одна или несколько переменных (возможно, различных типов), которых для удобства работы с ними сгруппировали под одним именем. Традиционный пример структуры – анкетные сведения о человеке: год рождения, место проживания, № телефона и т.д.

Представление сведений о воздушном объекте в виде структуры, в которой заключены данные о дальности, азимуте, высоте полета:

**struct target{**

**int d;**

**int b;**

**float h;**

**};**

Объявление структуры определяет тип данных. За правой фигурной скобкой, закрывающей список элементов, могут следовать переменные этого типа данных.

Например: **struct target {…} jet, bomb;**

Объявление структуры приводит к выделению памяти определенного размера. Если объявить структуру, не содержащую списка переменных, то память не резервируется, а просто описывает шаблон структуры. Если шаблон задан, то описать структурную переменную можно короче, например:

**struct target alt**;

что означает объявление переменной **alt** типа **struct target**.

При определении структуры ее можно инициализировать в виде списка константных выражений, например:

**struct target alt={250, 30, 2.5};**

запись означает, что в структуре **alt** переменной **d** (дальность) будет присвоено значение 250, **b** (азимут) 30, а **h** (высота) 2,5.

Чтобы обратиться к элементу структуры необходимо указать имя структуры и через точку – имя элемента данной структуры. Например, чтобы вывести на экран значение дальности до воздушного объекта, необходимо записать **cout<<alt.d;**Точка (**.**) означает оператор доступа к элементам структуры и соединяет имя структуры и имя её элемента.

**Структуры и функции**

Структуры могут передаваться функциям в качестве аргументов. В функцию можно передавать:

а) компоненты структуры по отдельности;

б) всю структуру целиком;

в) указатель на структуру.

Возвращать функции могут отдельную переменную структуры или всю структуру целиком.

Ниже представлен пример, в котором функция используется для инициализации переменных структуры.

**#include<iostream>**

**struct target** /\* Вначале на внешнем уровне описывается

**{**

**int d;**// шаблон структуры типа target \*/

**int b;**

**float h;**

**};**

**struct target init()** // Затем определяется функция init( ),

**{**

**struct target temp;**// которой не передаются аргументы,

**cout<< “Insertd:”;**// но которая возвращает структуру типа

**cin>>temp.d;** // target, предварительно заполнив ее поля

**cout<<“Insert b:”;**

**cin>>temp.b;**

**cout<<“Insert h:”;**

**cin>>temp.h;**

**return temp;**

**}**

**int main( )**

**{**

// В главной функции объявляется структура jet

**struct target**// типа target, в которую копируется структура,

**jet = init();**// возвращаемая функцией init( ).

**cout<<jet.d<<endl;** // Поля структуры выводятся на экран.

**cout<<jet.b<<endl;**

**cout<<jet.h<<endl;**

**return 0;**

**}**

Если функции передается большая структура, то чем копировать ее целиком, эффективней передать указатель на нее. Указатели на структуры ничем не отличаются от указателей на обычные переменные. Объявление

**struct target\*рtr;**

сообщает, что **рtr** – это указатель на структуру типа **struct target**. Если **рtr** указывает на структуру **target**, то **\*ptr** – это сама структура, а **(\*рtr).d** – ее элемент.

Указатели на структуры используются весьма часто, поэтому для доступа к ее элементам через указатель существует другая форма записи. Если **рtr**- указатель, то к той же переменной **d** обращаются так:

**рtr->d**

В следующем примере создана функция, которая получает указатель на структурную переменную **target jet** и обнуляет значения ее полей.

. . .

**struct target jet, \*р;** // объявление указателя

**р =&jet;**

**del (р);** // указателю **р** присваивается адрес структуры **jet** и

. . . // передается в функцию**del (р).**

**void del (struct target\* рh)**

**{**

**ph->d=0;**// если после работы функции del ( ) вывести

**ph->b=0;**// на экран поля структуры jet, то

**ph->h=0;** // они будут нулевыми

**}**

**Работа с файлами и форматирование данных**

**Форматирование данных с помощью манипуляторов**

Одним из способов форматирования информации в С++ является использование манипуляторов ввода/вывода.

Манипулятор – это ключевое слово языка С++, которое применяется в выражениях ввода/вывода для управления информацией ввода/вывода. Манипуляторы различают с параметрами и без параметров. Например:

**oct** – установка флага oct (вывод информации в восьмеричной форме) – манипулятор без параметра;

**setw(15)** – задание ширины поля вывода в 15 позиции – манипулятор с параметром.

Для того чтобы в программе можно было использовать манипуляторы с параметрами, необходимо в нее включить заголовок **<iomanip>**. Некоторые манипуляторы приведены в следующей таблице.

|  |  |  |
| --- | --- | --- |
| endl | Вывод символа новой строки и очистка потока | вывод |
| left | Установка флага left | вывод |
| right | Установка флага right | вывод |
| setprecision (int p) | Задание числа цифр точности, равной **р** | вывод |
| setw(int w) | Задание ширины поля вывода в **w** позиций | вывод |
| skipws | Отбрасываются начальные невидимые символы (пробелы, табуляции и символы новой строки) | ввод |
| uppercase | Вывод символов в верхнем регистре | вывод |
| ws | Пропуск начальных пробелов | ввод |
| setfill (intch) | Задание символа заполнения | вывод |

Пример с использованием манипуляторов.

**#include <iostream>**

**#include <iomanip>**

**int main()**

**{**

**cout<<hex <<100 <<endl;**

**cout<<oct<<10 <<endl;**

**cout<<setfill(‘Х’)<<setw(10)<<100;**

**cout<<’ ‘<<”Red nose.”<<endl;**

**return 0;**

**}**

Первый манипулятор **hex** сообщает потоку, что необходимо выводить целые числа в шестнадцатеричной системе счисления и выводит 100 в этой системе счисления. Второй манипулятор **oct** сообщает, что следующий вывод данных необходимо сделать в восьмеричной СС, и выводит 10 в восьмеричной СС. В следующей строке манипулятор **setfill(‘Х’)** предписывает заполнять пустые места символом **‘Х’**, а **setw(10)** устанавливает ширину поля вывода, равную десяти, в которую выводится число **100** и строка **Red nose.**

Результат выполнения программы:

64

12

ХХХХХХХ144 Red nose.

**Организация работы с файлами**

Для реализации файлового ввода/вывода в программу необходимо включить директиву: **#include<fstream>.**

При работе с файлами следует соблюдать следующий порядок.

1. Вначале необходимо создать поток для работы с файлом. Для этого объявляется его имя с соответствующим типом потока. Например:

**ifstream in;** - создан поток ввода из файла с именем in.

2) После создания потока, необходимо установить связь файла с потоком. Это производится с помощью функции open().

Например, для того чтобы связать созданный поток **in** с файлом, допустим с именем **one.txt**, необходимо записать:

**in.open(“one.txt”);** - это означает, что файл **one** связан с потоком **in** (для ввода данных).

3) Сделать проверку открытия файла:

**. . .**

**if(!in)**

**cout<<”Error of file opening”;**

4) После того, как файл открыт, работа с ним производится с помощью операторов ввода/вывода: “<<” и “>>”, так же, как и со стандартными потоками **cin** и **cout**, только вместо этих потоков необходимо использовать созданные потоки. Для нашего примера ввод из файла будет обеспечиваться инструкцией:

**in>> ...;**

*Вся информация в файле хранится в том же формате, как если бы она находилась на экране.*

5) После работы с файлом его необходимо закрыть с помощью функции close(). Для нашего примера это запишется:

**in.close();**

**Пример** программы работы с файлами, в которой создается файл для вывода и в него записываются данные, после чего он закрывается. Затем файл открывается для ввода и оттуда считываются данные в заранее подготовленные переменные.

**#include<iostream>**

**#include<fstream>**

**#include<iomanip>**

**using namespace std;**

**int main()**

**{**

**ofstream tofile;** //создание потока вывода

**tofile.open(“first”);** //открытие файла

**if(!tofile)** //проверка открытия файла

**cout<<”Error of file opening”;**

**else**

**{**

**tofile<<”Attention!\n”;**

**tofile<<100<<endl;**

**tofile.close();**

**}**

//открытие файла для ввода

**ifstream fromfile;** //создание потока ввода

**fromfile.open(“first”);** //открытие файла

**if(!fromfile)** //проверка открытия

**cout<<”Error of file opening”;**

**else**

**{**

**char str[20]; //** объявление переменных для ввода

**int num;** // данных из файла

**fromfile>>str>>n;**

**cout<<str<<’ \n‘<<num<<endl;**

**fromfile.close();**

**}**

**return 0;**

**}**

После завершения работы программы содержимое файла **first** будет следующим:

Attention!

100

**Практические задания**

*Выберите один из вариантов задания.*

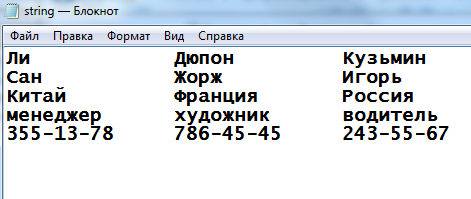
*Каждый пункт задания выполнить в виде отдельной функции.*

**Вариант А.**

**Задание 1.** Согласно индивидуальному номеру (см. таблица 4), используя *манипуляторы ввода-вывода, в цикле* сформировать матрицу на экране (без пробелов между элементами).

**Задание 2.** Разработать функцию преобразования *исходной строки* (согласно индивидуальному номеру, см. таблица 5), используя *только* библиотечные функции (обращение к элементам символьного массива по индексу запрещено). При этом для исходной строки *динамически выделить память в главной функции*.

**Задание 3.** Спроектировать структуру, описывающую характеристики «*Предмета»* (см. таблица 5). Создать массив из 5-ти структурных переменных и записать их в файл, как показано на рисунке ниже. Определить *Вычисляемый показатель*.



**Задание 4.** Разработать меню, в котором по желанию пользователя:

1. выполнить любой пункт задания;
2. прочитать из файла информацию о любом из «предметов»;
3. выйти из программы.

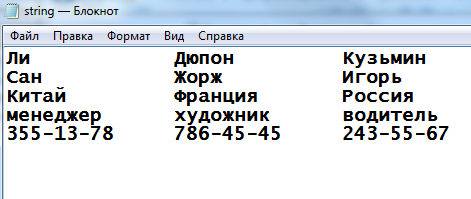
**Вариант В.**

*Каждый пункт задания выполнить в виде отдельной функции.*

**Задание 1.** Согласно индивидуальному номеру (см. таблица 4), используя *манипуляторы ввода-вывода, в цикле* сформировать матрицу на экране (без пробелов между элементами).

**Задание 2.** Разработать функцию преобразования *исходной строки* (согласно индивидуальному номеру, см. таблица 5), используя *только* библиотечные функции (обращение к элементам символьного массива по индексу запрещено). При этом для исходной строки *динамически выделить память в главной функции*.

**Задание 3.** Спроектировать структуру, описывающую характеристики «*Предмета»* (см. таблица 5). Создать массив из 3-х структурных переменных и записать их в файл, как показано на примере ниже. Определить *Вычисляемый показатель*.



**Вариант С.**

*Каждый пункт задания выполнить в виде отдельной функции.*

**Задание 1.** Согласно индивидуальному номеру (см. таблица 4), используя *манипуляторы ввода-вывода, в цикле* сформировать матрицу на экране (без пробелов между элементами).

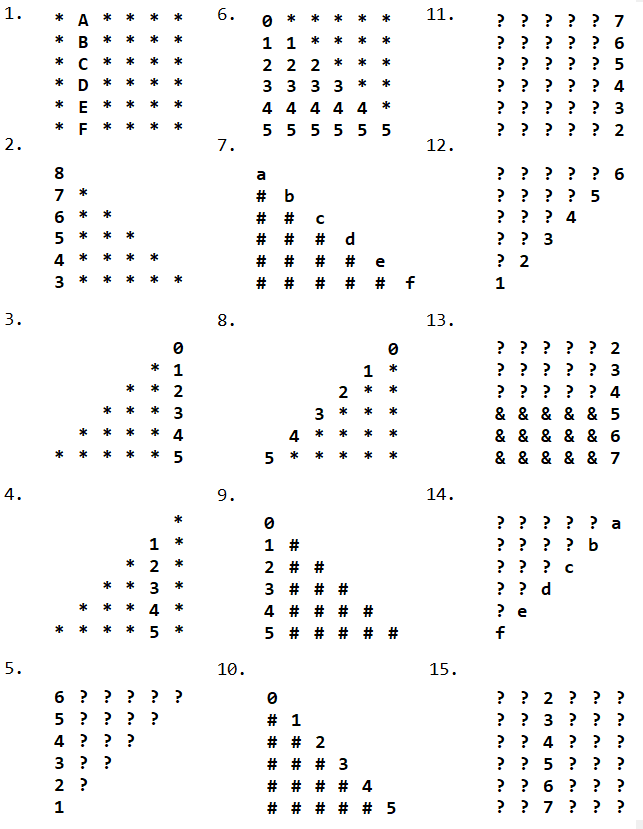
**Задание 2. Р**азработать функцию преобразования *исходной строки* (см. таблица 5), для которой *динамически выделить память*.

**Задание 3.** Спроектировать структуру, описывающую характеристики «*Предмета»* (см. таблица 5). Создать несколько структурных переменных и записать их в файл. Определить *Вычисляемый показатель*.

Отчет оформляется по общеустановленным правилам в *электронном виде* со следующим содержанием:

1. титульный лист,
2. тема и цель практической работы,
3. задание на практическую работу,
4. текст программы с комментариями,
5. результаты работы программы (вид экрана) и
6. выводы по созданному проекту и использованию средств языка программирования.

Таблица 4



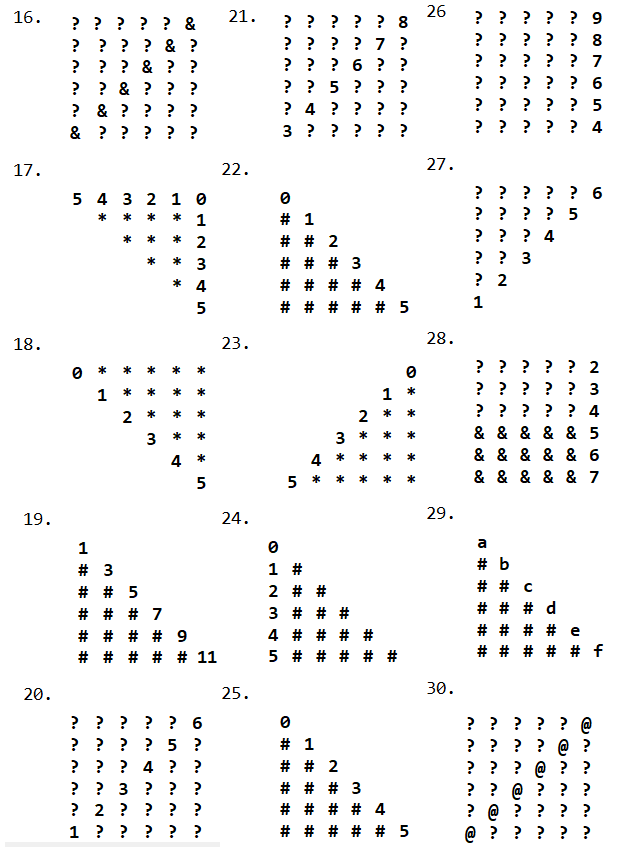


Таблица 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№**  **п/п //** | **Задание 2** | | | **Задание 3** | |
| **Исходная строка** | **Тип строки** | **Преобразование** | **Предмет** | **Вычисляемый показатель** |
|  | «Zero One Null» | Символьный массив | Заменить в строке все буквы «e» на «o» | Компьютер | Экземпляр с наибольшей оперативной памятью |
|  | «Zero Two Null» | Поменять местами Two и Null | Локальная сеть | Минимальная стоимость монтажа |
|  | «Zero Three Null» | Определить в строке количество букв «e» | Транспортное средство | Максимальный пробег на полном бензобаке |
|  | «Zero Four Null» | Определить местоположение в строке первой буквы «u» | Программный продукт | Последняя версия |
|  | «Zero Five Null» | Изменить порядок букв в Zeroна противоположный | Документ | Количество документов, выданных в прошлом году |
|  | «Zero Six Null» | Изменить каждую букву строки на букву, стоящую по алфавиту на две позиции левее | Диета | Максимальное дневное количество белков |
|  | «Zero Seven Null» | Добавить в строку слово One после Zero | Периферийное устройство компьютера | Минимальная цена устройства |
|  | «Zero Eight Null» | Исключить из строки пробелы | Строительный товар | Сумма покупки |
|  | «Zero Nine Null» | Поменять местами Null и Zero | Представитель университета | Количество студентов, обучающихся у конкретного преподавателя |
|  | «One Null Zero» | Изменить порядок букв в Nullна противоположный | Предприятие малого бизнеса | Название предприятия с максимальным числом сотрудников |
|  | «One One Zero» | Тип string | Определить местоположение первой буквы по алфавиту | СУБД | Количество СУБД заданного производителя |
|  | «One Two Zero» | Заменить все строчные буквы на заглавные | Страховой полис | Количество полисов на заданную фамилию |
|  | «One Three Zero» | Добавить слово Two после слова One | Наушники | Тип с максимальным частотным диапазоном |
|  | «One Four Zero» | Заменить в строке все буквы «o» на «E» | Недвижимость | Максимальная жилая площадь |
|  | «One Five Zero» | Определить в строке количество букв «о» | Часы | Самый дорогой экземпляр |
|  | «One Six Zero» | Заменить в строке все буквы «e» на «o» | Телефон | Самая новая модель |
|  | «One Seven Zero» | Поменять местами One и Zero | Бытовая техника | Количество товаров заданной фирмы |
|  | «One Eight Zero» | Определить в строке количество букв «e» | Магнитная карта для проезда на транспорте | Количество карт без поездок |
|  | «One Nine Zero» | Определить местоположение в строке первой буквы «e» | Насекомое | Максимальный размер крыльев |
|  | «Two Null Zero» | Изменить порядок букв в Zeroна противоположный | Канцелярские товары | Количество товаров заданной фирмы |
|  | «Two One Zero» | Символьный массив | Изменить каждую букву строки на букву, стоящую по алфавиту на четыре позиции левее | Товар | Сумма покупки |
|  | «Two Two Zero» | Заменить в строке слово Zero на слово Two | Учащийся | Учащийся с максимальным IQ (коэффициентом интеллекта) |
|  | «Two Three Zero» | Исключить из строки пробелы | Путешествие (тур) | Самый дешевый тур на 7 и более дней |
|  | «Two Four Zero» | Исключить из строки все буквы «o» | Мебель | Количество предметов из дерева |
|  | «Two Five Zero» | Вставить вместо пробелов знак подчеркивания | Программное обеспечение | Продукт с максимальным объемом памяти |
|  | «Two Six Zero» | Определить местоположение первой буквы по алфавиту | Представление | Минимальное число зрителей в зале |
|  | «Two Seven Zero» | Изменить порядок букв в Sevenна противоположный | Запоминающее устройство | Экземпляр с минимальным размером |
|  | «Two Eight Zero» | Тип string | Добавить слово One после слова Eight | Принтер | Экземпляр с наибольшей производительностью |
|  | «Two Nine Zero» | Заменить в строке все буквы «e» на «O» | Книга | Количество книг одного автора |
|  | «Three Null Zero» | Определить в строке количество букв «e» | Телефон | Самая новая модель |