ЛАБОРАТОРНАЯ РАБОТА № 22
БИБЛИОТЕКА ВВОДА-ВЫВОДА В ЯЗЫКЕ С++

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Ввод/вывод информации для консольных приложений в языке С++ организован с помощью библиотеки **iostream**, которая содержит классы, функции и перегруженные операторы, позволяющие проводить потоковый ввод/вывод информации, ее форматирование и т.д. Библиотека классов **iostream** – альтернатива функциям **printf** и **scanf** языка С.

Система ввода/вывода действует через потоки ввода/вывода. Потоки связаны с физическими устройствами:

cin – поток, связанный со стандартным устройством ввода (клавиатура);

cout – поток, связанный со стандартным устройством вывода (дисплей).

Основное достоинство библиотеки **iostream** – перегруженные операции, обеспечивающие удобную форму ввода-вывода данных.

Операция сдвига влево (**<<**) перегружена для обозначения вывода в поток и называется *операцией поместить в поток*.

Операция сдвига вправо (**>>**) перегружена для обозначения ввода потока и называется *операцией взять из потока*. Эти операции применяются к объектам стандартных потоков **cin** и **cout**.

Вывод данных

Вывод информации на дисплей может быть выполнен с помощью операции *поместить в поток*, т.е. перегруженной операции **<<.** Операция **<<** перегружена для вывода данных стандартных типов, а также для вывода строк и значений указателей.

```
int k = 123;
float f = 3.1415;
cout << k;
cout << f;
```

Строка **cout << k** выводит в стандартный поток вывода (на дисплей) значение переменной **k** целого типа, а строка **cout << f** – значение переменной **f** вещественного типа.

На дисплей будет выведены те же значения, которые были бы выведены, если бы мы использовали функцию **printf**:

```
printf("%d", k);
printf("%f", f);
```

Аналогично можно вывести на дисплей строки:

```
cout << "Hello!";
char Str[] = "Пример строки";
cout << Str;
```

С помощью функции **printf** эти же строки придется выводить так:

```
printf("Hello!");
char Str[] = "Пример строки";
printf("%s", Str);
```

Перегруженные операции *автоматически определяют тип переменных*, а в функции **printf** приходится использовать управляющую строку и спецификаторы форматов (**%d**, **%f**, **%s** и т.п.). Использование перегруженных операций из библиотеки **iostream** гораздо проще и удобнее.

Все перегруженные операции вставки в поток определены с типом возвращаемого значения **ostream&**.

Прототип имеет следующую форму:

```
ostream& operator<<(type);
```

Это позволяет выполнять конкатенацию вывода (сцепление строк), используя вставку в поток, например:

```
int k = 123;
float f = 3.1415;
cout << "k = " << k << ", f = " << f << "\n";
```

На дисплей последовательно будут выведено:

– строка **"k = "**
– значение переменной **k** (**123**)
– строка **", f = "**
– значение переменной **f** (**3.1415**)
– переход на новую строку

```
k = 123, f = 3.1415
```

С помощью функции **printf** эту же информацию придется выводить так:

```
printf("k = %d, f = %f\n", k, f);
```

Имя объекта стандартного потока вывода (**cout**) определено в библиотеке **iostream** в **пространстве имен std**.

Пространство имен – это декларативная программная область, в которой определяются различные идентификаторы (имена типов, функций, переменных и др.). Пространства имен используются для организации кода в виде программных групп, чтобы не допускать конфликтов имен, которые могут возникнуть, если в программу включается несколько библиотек.

Все идентификаторы в пределах пространства имен доступны друг другу. Вне пространства имен можно обращаться к идентификаторам, объявленным в пространстве, с помощью **полного имени** для каждого идентификатора (например **std::cout**), а также с помощью директивы **using** для одного идентификатора (**using std**::c**out**) или для всех идентификаторов сразу (**using namespace std**).

Поэтому в реальной программе придется указать «полное» имя потока:

```
std::cout << k;
std::cout << f;
```

или в начале программы написать:

```
using namespace std;
```

Кроме того, нужно подключить заголовочный файл с библиотекой **iostream**:

```
#include <iostream>
```

Пример программы, которая выводит на дисплей значения синуса в некотором диапазоне с заданным шагом, будет выглядеть так:

```
#include <iostream>
#include <locale.h>
#include <math.h>

using namespace std;
int main()
{
  float x = 0;

  setlocale(LC_ALL, "Russian"); // Смена кодировки
  cout << "Иванов Иван, 1 курс, группа 1\n";
  cout << "Лабораторная работа №1, вар. 25\n";
  for (int i = 0; i < 10; i++)
  {
    cout << "sin(" << x << ") = " << sin(x) << "\n";
    x += 0.1;
  }
}
```

```
Иванов Иван, 1 курс, группа 1
Лабораторная работа №1, вар. 25
sin(0) = 0
sin(0.1) = 0.0998334
sin(0.2) = 0.198669
sin(0.3) = 0.29552
sin(0.4) = 0.389418
sin(0.5) = 0.479426
sin(0.6) = 0.564642
sin(0.7) = 0.644218
sin(0.8) = 0.717356
sin(0.9) = 0.783327
```

Ввод данных

Операция ввода потока может быть выполнена с помощью операции *взять из потока*, т.е. перегруженной операции **>>.** Эта операция обычно игнорирует во входном потоке так называемые символы разделители или пробельные символы (пробелы, знаки табуляции, знак новой строки).

Операция **>>** перегружена для ввода данных стандартных типов, в том числе отдельных символов и строк.

```
int k;
float f;
cin >> k;
cin >> f;
```

Строка **cin >> k** вводит из стандартного потока ввода (с клавиатуры) значение переменной **k** целого типа, а строка **cin >> f** – значение переменной **f** вещественного типа. Значения вводятся до первого неправильно введенного символа. Перегруженные операции **>>** автоматически определяют тип переменных.

В языке С для ввода значений использовалась функция **scanf**. В отличие от операции ввода **>>** функция **scanf** не способна распознать тип вводимого значения. Так же, как и в функции **printf**, для обозначения типов вводимых данных приходится использовать управляющую строку и спецификаторы форматов (**%d**, **%f**, **%s** и т.п.):

```
scanf("%d", &k);
scanf("%f", &f);
```

Все перегруженные операции извлечения из потока определены с типом возвращаемого значения **istream&** (ссылка на объект класса).

Прототип имеет следующую форму:

```
istream& operator>>(type &);
```

Это позволяет выполнять конкатенацию ввода, например, для последовательного ввода целого и вещественного чисел:

```
int k;
float f;
cin >> k >> f;
```

При вводе данных с клавиатуры в этом примере значения для такого оператора должны быть разделены символами пробел, табуляция или Enter.

Имя объекта стандартного потока ввода (**cin**) также определено в библиотеке **iostream** в пространстве имен **std**.

```
#include <iostream>
#include <locale.h>
#include <math.h>

using namespace std;
int main()
{
  int a, b;

  setlocale(LC_ALL, "Russian"); // Смена кодировки
  cout << "Иванов Иван, 1 курс, группа 1\n";
  cout << "Лабораторная работа №1, вар. 25\n\n";
 cout << "Введите два целых числа: ";
 cin >> a >> b;
  cout << "Результат: " << a << " + " << b << " = " << a+b << "\n";
}
```

```
Иванов Иван, 1 курс, группа 1
Лабораторная работа №1, вар. 25

Введите два целых числа : 4 7
Результат: 4 + 7 = 11
```

Специфика ввода символьных строк: по умолчанию потоковый ввод **cin** вводит строку до пробела, символа табуляции или перевода строки. Для ввода текста до символа перевода строки используется функция-член класса **getline()** (см. далее).

Манипуляторы потоков

В цепочку операций вставки в поток или извлечения из потока можно вставлять так называемые манипуляторы потоков, которые решают задачи форматирования. Манипуляторы потоков позволяют задавать основание системы счисления, ширину полей и точность, вставку в поток символа новой строки и т.д. Кроме манипуляторов можно использовать функции-члены класса, определенные в библиотеке **iostream**, которые тоже позволяют задавать ширину полей, точность и другие параметры.

Установка основания системы счисления

Манипуляторы потоков **dec**, **oct** и **hex** позволяют задавать основание системы счисления как при выводе, так и при вводе целых чисел (соответственно **10**, **8** и **16**).

```
int a, b;

cout << "--- 10-ная система счисления ---" << "\n";
cout << "Введите два целых числа : ";
cin >> a >> b;
cout << "Результат: " << a << " + " << b << " = " << a + b << "\n";

cout << "--- 16-ная система счисления ---" << "\n";
cout << hex;
cout << "Введите два целых числа : ";
cin >> a >> b;
cout << "Результат: " << a << " + " << b << " = " << a + b << "\n";

cout << "--- 8-ная система счисления ---" << "\n";
cout << oct;
cout << "Введите два целых числа : ";
cin >> a >> b;
cout << "Результат: " << a << " + " << b << " = " << a + b << "\n";
```

```
--- 10-ная система счисления ---
Введите два целых числа : 20 30
Результат: 20 + 30 = 50
--- 16-ная система счисления ---
Введите два целых числа : 20 30
Результат: 14 + 1e = 32
--- 8-ная система счисления ---
Введите два целых числа : 20 30
Результат: 24 + 36 = 62
```

Манипулятор потока **scientific** используется при выводе на дисплей вещественных чисел в виде мантиссы с порядком (научная нотация), а **fixed** – при выводе вещественных чисел в формате с фиксированной точкой:

```
float f;
cout << "Введите вещественное число: ";
cin >> f;
cout << "Результат: " << scientific << f << "\n";
cout << "Результат: " << fixed << f << "\n";
```

```
Введите вещественное число: 123.456703
Результат: 1.234567e+02
Результат: 123.456703
```

Установка ширины поля вывода

Для размещения чисел различной длины в полях постоянной ширины можно воспользоваться манипулятором **setw(int)**:

```
cout << setw(10) << f << "\n";
```

или функцией-членом **width(int)**:

```
cout.width(10);
cout << f << "\n";
```

Они устанавливают ширину поля, выравнивая значения выводимых чисел по правому краю соответствующих полей. Поля вывода дополняются пробелами. При выравнивании по правому краю пробелы вставляются слева от значений. Символ, используемый для дополнения поля, называется символом-заполнителем. По умолчанию устанавливается выравнивание по правому краю, а символ-заполнитель – пробел.

```
float f = 123.456789;
cout << "f = " << setw(10) << f << "\n";
cout << "f = " << setw(12) << f << "\n";
cout << "f = " << setw(14) << f << "\n";
cout << "f = " << setw(16) << f << "\n";
```

```
f =    123.457
f =      123.457
f =        123.457
f =          123.457
```

По умолчанию **cout** заполняет неиспользуемые части поля пробелами. Для изменения этого, можно воспользоваться функцией-членом **fill(char)**. Например, следующий вызов изменяет символ-заполнитель на точку:

```
float f = 123.456789;
cout.fill('.');
cout << "f = " << setw(10) << f << "\n";
cout << "f = " << setw(12) << f << "\n";
cout << "f = " << setw(14) << f << "\n";
cout << "f = " << setw(16) << f << "\n";
```

```
f = ...123.457
f = .....123.457
f = .......123.457
f = .........123.457
```

Ширина поля устанавливается только для следующего отображаемого элемента, после чего она возвращается к значению по умолчанию. В следующем примере

```
float f = 123.456789;
cout << "f = " << setw(10) << f << ", f = " << f << "\n";
```

при первом выводе числа **f** будет выделено 10 знакомест, а при втором – нет.

```
f =    123.457, f = 123.457
```

В отличие от ширины поля, новый символ-заполнитель остается в действии до тех пор, пока он не будет заменен.

Установка точности отображения чисел с плавающей точкой

Смысл точности чисел плавающей точкой зависит от режима вывода. В режиме по умолчанию она означает общее количество отображаемых разрядов. В фиксированной и научной нотации, точность означает количество десятичных разрядов, отображаемых справа от точки. Точность по умолчанию, применяемая в C++, обычно равна шести (следует помнить, что завершающие нули отбрасываются.) Функция-член **precision(int)** и манипулятор **setprecision(int)** позволяет выбрать другие значения. Например, следующий оператор устанавливает точность вывода **cout** в **2**:

```
cout.precision(2);
```

В отличие от **width(int)**, новое значение точности остается в действии до тех пор, пока не будет переустановлено.

Примеры вывода в разных режимах с различной точностью:

```
// вывод в режиме по умолчанию
cout << setprecision(3) << f << "\n";
cout << setprecision(4) << f << "\n";
cout << setprecision(5) << f << "\n";
cout << setprecision(6) << f << "\n";
```

```
123
123.5
123.46
123.457
```

```
// вывод в научной нотации
cout << scientific;
cout << setprecision(3) << f << "\n";
cout << setprecision(4) << f << "\n";
cout << setprecision(5) << f << "\n";
cout << setprecision(6) << f << "\n";
```

```
1.235e+02
1.2346e+02
1.23457e+02
1.234568e+02
```

```
// вывод в формате с фиксированной точкой
cout << fixed;
cout << setprecision(3) << f << "\n";
cout << setprecision(4) << f << "\n";
cout << setprecision(5) << f << "\n";
cout << setprecision(6) << f << "\n";
```

```
123.457
123.4568
123.45679
123.456787
```

Если в программе используются манипуляторы **setprecision** и **setw** , необходимо подключить заголовочный файл `#include <iomanip>` .

Кроме перечисленных манипуляторов используются и другие, например:

`endl` – помещение в выходной поток символа конца строки **'\n'**;

`showpos` – отображать знак + для положительных чисел;

`left` — выравнивать по левому краю поля;

`right` — выравнивать по правому краю поля;

`uppercase` – при выводе использовать символы верхнего регистра

и другие.

Часто используются и функции-члены, например `get()` – ожидает ввод символа, а `getline(указатель, количество)` – позволяет ввод текста до символа перевода строки, максимальное количество символов ограничено полем *количество:*

```
char s[80], c;
cin.getline(s, 80);
cout << s << endl;
c = cin.get();
cout << c << endl;
```

Функция **setf(int)** предоставляет дополнительные средства настройки потоков с помощью битовых флагов.

Пример 1. *Форматированный вывод в таблицу*

В лабораторные работы 19 и 20 были посвящены массивам структур. В программе создавался массив структур, вводились их параметры помощью функции scanf, вычислялись новые параметры, результаты выводились на дисплей с помощью функции print и т.д.

Переделаем программу под потоки ввода/вывода. Для этого добавим нужные заголовочные файлы и переделаем ввод данных на поток **cin**, а вывод – на **cout**. В программе будем использовать структуры типа Ромб (struct Rhombus), добавим возможность задавать размер массива ромбов и создавать его динамически, заполнять его случайными числами, вычислять площади и периметры и выводить результаты в виде таблицы. Для выравнивания данных в таблице будем использовать соответствующие манипуляторы.

```
#include <iostream>
#include <iomanip>
#include <locale.h>
```

```
using namespace std;

struct Rhombus // Ромб
{
  int x;  // Координата x центра ромба
  int y;  // Координата y центра ромба
  int d1; // Первая диагональ ромба
  int d2; // Вторая диагональ ромба
};

void FillArrayRand(Rhombus* A, int n)  // заполнение массива
{                    // случайными числами
  int i;
  for (i = 0; i < n; i++)
  {
    A[i].x = (float)rand() / RAND_MAX * 30 + 1;
    A[i].y = (float)rand() / RAND_MAX * 30 + 1;
    A[i].d1 = (float)rand() / RAND_MAX * 30 + 2;
    A[i].d2 = (float)rand() / RAND_MAX * 30 + 3;
  }
}

float RArea(Rhombus a) // Вычисление площади ромба
{
  return (float)a.d1 * a.d2 / 2;
}

float RPerimeter(Rhombus a) // Вычисление периметра ромба
{
  return 2 * sqrt((float)a.d1 * a.d1 + a.d2 * a.d2);
}

void ViewArrTabl(Rhombus* A, int n)
{
  cout.precision(1);
  cout << "-------------------------------------------------\n";
  cout << "| N | x | y | d1 | d2 | Площадь | Периметр |\n";
  cout << "|----|-----|-----|-----|-----|---------|----------|\n";
  for (int i = 0; i < n; i++)
  {
    cout << "| " << setw(2) << i + 1;
    cout << " |" << setw(4) << A[i].x
      << " |" << setw(4) << A[i].y
      << " |" << setw(4) << A[i].d1
      << " |" << setw(4) << A[i].d2
      << " |" << fixed << setw(8) << RArea(A[i])
      << " |" << setw(8) << RPerimeter(A[i]) << " |" << endl;
  }
  cout << "-------------------------------------------------\n";
}

int main()
{
  Rhombus* ArrR;
  int size;
```

```
  setlocale(0, "Russian");
  cout << "Размер массива ромбов: ";
  cin >> size;
  ArrR = new Rhombus[size];

  FillArrayRand(ArrR, size);
  cout << "============== Массив ромбов ==========\n";
  ViewArrTabl(ArrR, size);

  cin.get();
  return 0;
}
```

```
Размер массива ромбов: 11
============== Массив ромбов ==========
-------------------------------------------------
| N  |  x  |  y  |  d1 |  d2 | Площадь | Периметр |
|----|-----|-----|-----|-----|---------|----------|
|  1 |   1 |  17 |   7 |  27 |    94.5 |     55.8 |
|  2 |  18 |  15 |  12 |  29 |   174.0 |     62.8 |
|  3 |  25 |  23 |   7 |  28 |    98.0 |     57.7 |
|  4 |  22 |  16 |  11 |   3 |    16.5 |     22.8 |
|  5 |   3 |  11 |   6 |   7 |    21.0 |     18.4 |
|  6 |  30 |  14 |   5 |   3 |     7.5 |     11.7 |
|  7 |   1 |  12 |  17 |  20 |   170.0 |     52.5 |
|  8 |  19 |  19 |   6 |  22 |    66.0 |     45.6 |
|  9 |  14 |  11 |   3 |  21 |    31.5 |     42.4 |
| 10 |  24 |  25 |  17 |  12 |   102.0 |     41.6 |
| 11 |  27 |  22 |  30 |  30 |   450.0 |     84.9 |
-------------------------------------------------
```

Проблемы с очисткой буфера ввода

В лабораторной работе 17 описывались проблемы, которые возникают из-за того, что во входном буфере остаются ненужные символы. Эта проблема решалась с помощью функции getchar(), которую мы вызывали в цикле до тех пор, пока не встретится символ ‘\n’:

```
  while (getchar() != '\n');
```

При этом очищался входной буфер.

В библиотеке **iostream** есть специальная функция-член **ignore()**, которая решает эту же задачу. Функция принимает два аргумента: число, указывающее максимальное количество символов для чтения, и символ, служащий разделителем при вводе. Например, вызов функции

```
  cin.ignore(80, '\n');
```

считывает из входного буфера и отбрасывает первые 80 символов или все символы вплоть до символа перевода строки, в зависимости от того, что произойдет раньше. Поэтому, если в программе приходится выполнять последовательный ввод различной информации, то можно использовать эту функцию.

```
    cout << "Введите число ромбов: ";
    cin >> size;
    cin.ignore(80, '\n'); // Очистка входного буфера
```

Проблемы кодировки

Еще одна проблема проявляется при вводе и **отображении русских букв**. Функция setlocale(LC_ALL, “Russian”) работает только *на вывод* символов на дисплей, а

на ввод с клавиатуры – нет. Поэтому вместо функции setlocale() лучше использовать две функции: SetConsoleCP() и SetConsoleOutputCP().

Функция SetConsoleCP() устанавливает кодовую страницу **ввода**, используемую консолью, чтобы преобразовывать ввод информации с клавиатуры в соответствующие символьные значения.

Функция SetConsoleOutputCP() устанавливает кодовую страницу **вывода** данных, используемую консолью, чтобы преобразовать символьные значения, написанные различными функциями вывода информации в изображение, показываемое на экране в консольном окне.

Для установки кодовой страницы 1251 (с русскими буквами) нужно в начале функции main() вставить:

```
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
```

тогда и ввод, и вывод будут корректно работать. Не забудьте подключить заголовочный файл `#include <windows.h>`.

Пример 2. *Форматированный ввод/вывод*

В лабораторной работе 16 использовались структуры, содержащие не только числовые поля, но и строки. Напишем программу, использующую ту же структуру, но ввод и вывод выполним с помощью библиотеки **iostream**.

Добавим нужные заголовочные файлы и переделаем ввод данных на поток **cin**, а вывод – на **cout**. В программе будем использовать структуру **ManInfo** из лабораторной работы 17. Переделаем функции ввода параметров пациента и вывода результатов в таблицу. Для выравнивания данных в таблице будем использовать соответствующие манипуляторы.

Для решения проблем кодировки будем использовать функции **SetConsoleCP()** и **SetConsoleOutputCP()**, а для очистки буфера ввода от «лишних» символов – функцию **ignore()**.

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

using namespace std;

struct ManInfo
{
  char   Family[50]; // Фамилия
  short   Age;     // Возраст
  float   Weight;    // Вес
  float   Height;    // Рост
};

void SetInfo(ManInfo* pman) // Задает поля структуры
{               // по указателю pman
  cout << "Фамилия: ";
  cin.getline(pman->Family, 31);
  cout << "Возраст: ";
  cin >> pman->Age;
  cin.ignore(255, '\n'); // Очистка буфера
  cout << "Вес: ";
  cin >> pman->Weight;
  cin.ignore(255, '\n'); // Очистка буфера
  cout << "Рост: ";
```

```
  cin >> pman->Height;
  cin.ignore(255, '\n'); // Очистка буфера
}
void ViewInfo(ManInfo* Arr, int Len) // Вывод инф-и на дисплей
{
  cout.precision(1);
  cout << "=================================================\n";
  cout << "| N |   Фамилия    | Возраст | Вес | Рост |\n";
  cout << "|=================================================|\n";
  for (int i = 0; i < Len; i++)
  {
    cout << "| " << setw(2) << i + 1;
    cout << " | " << left << setw(18) << Arr[i].Family
      << "| " << right << setw(3) << Arr[i].Age
      << "  | " << fixed << setw(5) << Arr[i].Weight
      << " | " << setw(5) << Arr[i].Height << " |" << endl;
  }
  cout << "=================================================\n";
}
int main()
{
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);

  ManInfo* MArr;
  int size;

  cout << "Введите число пациентов: ";
  cin >> size;
  cin.ignore(255, '\n'); // Очистка буфера

  MArr = new ManInfo[size];
  for (int i = 0; i < size; i++)
  {
    cout << "Задайте данные " << i + 1 << " пациента:\n";
    SetInfo(&MArr[i]);
  }
  ViewInfo(MArr, size);

  cin.get();
  return 0;
}
```

```
Задайте данные 1 пациента:
Фамилия: Иванов А.Б.
Возраст: 45
Вес: 56
Рост: 167
Задайте данные 2 пациента:
Фамилия: Петров С.Д.
Возраст: 34
Вес: 56
Рост: 165
Задайте данные 3 пациента:
Фамилия: Сидоров Н.И.
Возраст: 56
Вес: 78
Рост: 181
=============================================================
|  N |        Фамилия        | Возраст |   Вес   |   Рост  |
|=============================================================|
|  1 | Иванов А.Б.           |   45    |    56.0 |  167.0  |
|  2 | Петров С.Д.           |   34    |    56.0 |  165.0  |
|  3 | Сидоров Н.И.          |   56    |    78.0 |  181.0  |
=============================================================
```

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

В начале программы ОБЯЗАТЕЛЬНО выводить:
ФИО, группа, номер лаб. работы, номер варианта.

Задание 1. Разработать программу для визуализации работы с массивами структур, используемых в лабораторных работах 19 и 20, с помощью **потоков ввода/вывода**. В программе необходимо задавать размер массива структур, динамически выделять под него память и заполнять массив случайными числами. Программа должна выводить данные структур, хранящихся в массиве, а также их площади, периметры, длины диагоналей и т.п. (в зависимости от используемой структуры) на дисплей **в виде таблицы**, используя соответствующие манипуляторы для форматирования данных в таблице.

В программе **не должны** использоваться функции **scanf**, **printf**, **getch** и т.п.

Для получения максимального балла:

– Заполнять массив тремя способами: случайными числами, вручную и по порядку от заданного начального значения с шагом k.

– Выводить в виде таблицы исходный массив и отсортированный массив. Для сортировки использовать функцию из лабораторной работы 20.

Задание 2. Разработать программу для визуализации работы с массивами структур, используемых в лабораторной работе 16, с помощью **потоков ввода/вывода**. В программе необходимо задавать размер массива структур, динамически выделять под него память и заполнять массив корректными данными. Программа должна выводить данные структур, хранящихся в массиве, на дисплей **в виде таблицы**, используя соответствующие манипуляторы для форматирования данных в таблице.

В программе **не должны** использоваться функции **scanf**, **printf**, **getch** и т.п.

Для получения максимального балла нужно выполнить следующие требования:

– программа должна проверять корректность вводимой с клавиатуры информации и не позволять вводить заведомо неверные данные;