ЛАБОРАТОРНАЯ РАБОТА № 13
ТИПИЧНЫЕ ЗАДАЧИ С МАССИВАМИ И АЛГОРИТМЫ ИХ РЕШЕНИЯ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Задачи по работе с массивами можно разбить на следующие типы:

1. Последовательная обработка элементов массива.
2. Выборочная обработка элементов массива.
3. Поиск в массиве единственного элемента, отвечающего некоторому условию.
4. Одновременная обработка нескольких массивов.
5. Изменение порядка следования элементов без изменения размеров исходного массива.
6. Переформирование массива с изменением его размеров.

Для каждого типа существуют свои приемы программирования. В реальных программах задачи в чистом виде встречаются довольно редко, но любую сложную задачу можно разделить на более простые задачи, относящиеся к указанным выше типам.

1. Последовательная обработка элементов массива

К этому типу задач относятся: нахождение максимального и минимального элементов массива, суммы элементов, произведения элементов, среднего арифметического, среднего геометрического, подсчет количества элементов, отвечающих определенному условию или обладающих некоторыми признаками, а также их суммы, произведения и т.д., кроме того, задачи ввода и вывода массивов, изменению элементов массива по некоторому правилу. Особенностью задач данного типа является то, что количество обрабатываемых элементов массива известно. Это позволяет использовать счетные циклы, параметр которых обеспечивает доступ к элементам.

Пример 1. *Нахождение максимального и минимального элементов массива, номеров первого (последнего) максимального (минимального) элементов массива*

Для решения подобных задач необходима переменная (`min`) для хранения найденного минимального значения и сам массив.

```
float Arr[N]; // массив, размером N
//  . . .   заполняем массив данными
float min; // для минимального значения
```

Сначала записываем в эту переменную значение первого элемента (`Arr[0]`). Затем берем следующий элемент и сравниваем его с минимальным. Если он меньше минимального, записываем его значение в переменную `min`, и так далее. После перебора всех элементов массива, **в переменной `min` будет минимальное значение** среди всех элементов массива. Перебор в цикле нужно начинать с элемента с номером 1 (а не 0), поскольку начальный элемент мы рассмотрели отдельно.

```
float min = Arr[0]; // предполагаем, что Arr[0] - минимальный
int i;
for (i=1; i < N; i ++) // цикл по всем элементам, начиная с Arr[1]
  if (Arr[i] < min)    // если Arr[i] меньше min
    min = Arr[i];     // запомнить значение Arr[i] как минимальное
```

Если нужно найти **номер** первого минимального элемента, объявляем переменную (`imin`), в которой будем запоминать этот номер. Тогда минимальное значение будет `Arr[imin]`.

```
int imin = 0;  // предполагаем, что Arr[0] - минимальный
int i;
for (i=1; i < N; i ++) // цикл по всем элементам, начиная с Arr[1]
  if (Arr[i] < Arr[imin]) // если Arr[i] меньше Arr[imin]
    imin = i;             // запомнить его номер
```

Если нужен номер последнего минимального элемента нужно сравнивать на `<=`:

```
int imin = 0;  // предполагаем, что Arr[0] - минимальный
int i;
for (i=1; i < N; i ++) // цикл по всем элементам с Arr[1]
  if (Arr[i] <= Arr[imin])  // если Arr[i] меньше Arr[imin]
    imin = i;       // запомнить его номер
```

Аналогично ищется максимальное значение**.**

Подобные задачи поиска лучше всего оформить в виде функций, которые можно было бы использовать в своих программах.

Создадим функцию поиска минимального элемента массива. Пусть имя функции будет **FindMin.** Определяем аргументы функции: во-первых, сам массив, в котором будем искать минимальный элемент, во-вторых, его длина. Для передачи массива в качестве параметра нужно в заголовке функции после формального имени массива указать пустые квадратные скобки:

```
float FindMin(float Arr[], int Size)
```

Можно также передать массив в функцию в виде указателя:

```
float FindMin(float *Arr, int Size)
```

Функция должна возвращать найденное значение, значит тип возвращаемого значения функции должен быть таким же, как тип элементов массива (в данном примере – `float`) .

```
float FindMin(float *Arr, int Size)
{
  float min = Arr[0];  // предполагаем, что A[0] - минимальный
  for (int i = 1; i < Size; i ++) // цикл по всем элементам
    if (Arr[i] < min)   // если Arr[i] меньше min
      min = Arr[i];  // запомнить значение Arr[i] как минимальное
  return min; //возвращаем найденное минимальное значение
}
```

Пример использования функции:

```
#define SIZE 100
int main()
{
  float Array[SIZE];
  float minValue;
//  ... Заполняем массив Array
//
  minValue = FindMin(Array, SIZE); // ищем минимальное значение
  printf("Минимальное значение в массиве = %f\n", minValue);
//...
  return 0;
}
```

Если нужно возвращать номер минимального значения, то тип возвращаемого значения функции должен быть **int** или лучше **unsigned int** (т.к. номер не может быть отрицательным).

Пример 2. *Подсчет количества элементов массива, суммы всех элементов массива, суммы элементов массива с заданными свойствами.*

Для решения подобных задач вводим переменную – счетчик, в которой будем подсчитывать количество нужных элементов. Для подсчета по нескольким параметрам вводим несколько счетчиков. Для подсчета суммы всех элементов необходима дополнительная переменная, в которой будем накапливать сумму. Например:

а) количество отрицательных элементов в массиве:

```
int counter = 0;    // счетчик
for (int i = 0; i < N; i++)     // перебор массива
  if (Arr[i] < 0)  // если отрицательный элемент
    counter++;
```

б) сумма всех элементов массива:

```
int summ = 0;    // для подсчета суммы
for (int i = 0; i < N; i++)
  summ += Arr[i];
```

В программе лучше создать функции для решения задач данного типа, например функция вычисления суммы всех элементов массива:

```
float SummArray(float *Arr, int Size)
{
  int summ = 0;    // для подсчета суммы
  for (int i = 1; i < Size; i ++) // цикл по всем элементам
    summ += Arr[i];
  return summ;
}
```

У функции, подсчитывающей количество элементов с заданными свойствами, тип возвращаемого значения должен быть **int** или лучше **unsigned int**.

Пример 3. *Нахождение среднего арифметического и среднего геометрического*

Среднее арифметическое ряда чисел – это сумма данных чисел, поделенная на количество слагаемых. Например, среднее всех положительных элементов массива:

```
  int counter = 0;    // счетчик
  int summ = 0;       // для подсчета суммы
  float aver;
  for (int i = 0; i < N; i++)
    if (Arr[i] > 0)
    {
      summ += Arr[i];
      counter++;
    }
  aver = summ / counter;
```

Среднее геометрическое ряда чисел – корень n-й степени из произведения этих чисел – ищется аналогично.

2. Выборочная обработка элементов массива

Задачи данного типа похожи на задачи первого типа, но обработка ведется не над всеми элементами массива, а только над теми, которые имеют вполне определенное значение индексов. С целью уменьшения времени работы программы и увеличения ее эффективности, программисту следует найти закон изменения индексов нужных элементов и обеспечить его исполнение. В зависимости от закона изменения индексов могут использоваться все виды циклов, а также их сочетание.

Пример 4. *Определить максимальный элемент, среди элементов целочисленного массива с четными индексами.*

При решении этой задачи нет смысла перебирать все элементы, нужно начать со 0-го элемента и увеличивать индекс на два после каждой проверки. Количество анализируемых элементов при этом уменьшается вдвое.

```
int max = Arr[0]; // предполагаем, что Arr[0] - максимальный
int i;
for (i=2; i < N; i+=2) // цикл по всем четным элементам с Arr[2]
  if (Arr[i] > max)     // если Arr[i] больше max
    max = Arr[i];   // запомнить значение Arr[i] как максимальное
```

Пример 5. *Подсчитать сумму элементов массива с индексами кратными 3.*

При решении этой задачи нет смысла перебирать все элементы, нужно начать со 0-го элемента и увеличивать индекс на три после каждого суммирования. Количество анализируемых элементов при этом уменьшается в три раза.

```
  int summ = 0;   // для подсчета суммы
  int i;
  for (i = 0; i < N; i += 3) // Каждый третий элемент
    summ += Arr[i];
```

3. Поиск в массиве единственного элемента, отвечающего некоторому условию

Примерами подобного рода задач могут служить поиск первого отрицательного, первого положительного и любого первого элемента, отвечающего некоторому условию, поиск первого или единственного элемента, равного некоторому конкретному значению. Особенность задач этого класса в том, что нет необходимости просматривать весь массив. Просмотр нужно закончить сразу, как только требуемый элемент будет найден. При этом может производиться как поэлементный просмотр, так и выборочная обработка массива. Однако, в худшем случае, для поиска элемента требуется просмотреть весь массив. Такой тип поиска называется линейным. При программировании поисковых задач используются циклы, в которых условие выхода формируется из двух условий. Одно условие – пока элемент не найден, а второе – пока есть элементы массива. После выхода из цикла осуществляется проверка, по какому из условий произошел выход.

Пример 6. *Определить, есть ли в массиве отрицательные элементы*

Для решения задачи удобно ввести специальную переменную (называется флагом), которая будет устанавливаться в единицу, если анализируемая гипотеза подтверждается, или останется равной нулю, если нет. В цикле проверяем все заданные элементы массива, по результатам устанавливается или сбрасывается флаг.

```
int flag = 0;   // флаг сброшен
for (int i = 0; i < N; i++)    // перебор массива
  if (Arr[i] < 0)  // если отрицательный элемент
  {
    flag = 1;// взводим флаг
    break; // и выходим из цикла, т.к. дальше проверять нет смысла
```

```
  }
// переменная flag показывает, есть ли отрицательные элементы
```

Функция, реализующая данный алгоритм, должна возвращать эту переменную **flag**, т.е. тип возвращаемого значения будет **int**.

```
int FindNegativ(float *Arr, int Size)
{
  int flag = 0;   // флаг сброшен
  for (int i = 0; i < Size; i++)    // перебор массива
    if (Arr[i] < 0)  // если отрицательный элемент
    {
      flag = 1;// взводим флаг
      break; // выходим из цикла, т.к. дальше проверять нет смысла
    }
  return flag;
}
```

Можно обойтись и без дополнительной переменной-флага, кроме того, в качестве возвращаемого значения можно использовать тип **bool**.

```
bool FindNegativ(float *Arr, int Size)
{
  for (int i = 0; i < Size; i++)    // перебор массива
    if (Arr[i] < 0)  // если отрицательный элемент
      return true; // выходим из цикла и функции, возвращая true
    }
  return false;
}
```

Пример 7. *Найти в целочисленном массиве первый элемент кратный 7*

При решении этой задачи просматриваем по очереди все элементы и проверяем условие (кратность семи). Если элемент найден, нет смысла продолжать цикл, выходим из него с помощью оператора **break**. Если же такой элемент в массиве отсутствует, цикл завершится, когда индекс массива станет равным размеру массива. Проверяя значение индекса можно определить, был ли найден нужный элемент.

```
  int i;
  for (i = 0; i < N; i++) // Цикл по всем элементам
    if (Arr[i] % 7 == 0) // Если значение элемента кратно 7
      break;             // Досрочно завершаем цикл

  // По завершению цикла:
  if (i == N) // Если просмотрели все элементы массива
    printf("Нет такого элемента\n");
  else
    printf("Первый элемент кратный семи = %d\n", Arr[i]);
```

Для оформления данного алгоритма в виде функции нужно решить вопрос о том, что эта функция должна возвращать. Если выводить информацию на дисплей в самой функции, то это ограничит ее применимость, поэтому возвращать функция должна флаг: найден элемент или нет. С другой стороны, нужно вернуть и само найденное значение.

Но с помощью оператора **return** можно вернуть только одно значение. Поэтому само найденное число придется возвращать через аргумент-указатель (запишем его в список аргументов функции как **int *elem**).

```
bool FindElement(int *Arr, int Size, int *elem)
{
  int summ = 0;    // для подсчета суммы
  int i;
  for (i = 0; i < Size; i++) // Цикл по всем элементам
    if (Arr[i] % 7 == 0) // Если значение элемента кратно 7
      break;             // Досрочно завершаем цикл

  // По завершению цикла:
  if (i == Size) // Если просмотрели все элементы массива
    return false;  // Элемент не найден
  *elem = Arr[i];  // Записываем по адресу elem найденное значение
  return true;  // Элемент найден
}
```

При вызове функции передаем в нее в качестве аргументов сам массив (**Array**), размер массива (**SIZE**), и адрес переменной **k** (**&k**), в которой будет сохраняться найденное число.

```
#define SIZE 100
int main()
{
  int Array[SIZE];
  int k;
//  ... Заполнение массива
  if (FindElement(Array, SIZE, &k)) // Передаем в функцию адрес k
    printf("Первый элемент кратный семи = %d\n", k); // Если нашли
  else
    printf("Нет такого элемента\n");
//...
  return 0;
}
```

Пример 8. *Разработать функцию, определяющую, есть ли в массиве знакочередующиеся элементы*

Если в массиве могут быть нулевые элементов, нужно прямо сравнивать знаки **i**-того и **i+1**-го элементов:

```
  if (Arr[i]>0 && Arr[i+1]<0 || Arr[i]<0 && Arr[i+1]>0)
```

Если же нулевых элементов нет, то условие может быть проще:

```
  if (Arr[i] * Arr[i+1] < 0)
```

Так как в условии проверяется **i+1**-ый элемент, условие завершения цикла должно быть **i < Size - 1**.

```
bool FindAlternative(float *Arr, int Size)
{
  for (int i = 0; i < Size - 1; i++)    // перебор массива
    if (Arr[i] > 0 && Arr[i+1] < 0 || Arr[i] < 0 && Arr[i+1] > 0)
      return true; // выходим из цикла и функции, возвращая true
    }
  return false;
}
```

4. Одновременная обработка нескольких массивов

К этому типу относятся задачи по копированию всего или части одного массива в другой, объединению массивов, копировании элементов с заданными свойствами из одного массива в другой и т.д.

Особенностью задач этого типа является то, что при работе с несколькими массивами, у каждого из них должен быть свой индекс и свой способ заполнения. После копирования число элементов в массивах тоже будет разным.

Пример 9. *Скопировать из одного массива в другой четные числа*

Для каждого массива нужно использовать свою переменную индексирования (например, **i** и **k**). Элементы исходного массива просматриваем по очереди, если условие истинно, копируем элемент в другой массив по его индексу (**k**).

```
#define SIZE 100

int ArrayA[SIZE], ArrayB[SIZE];
int i, k;
for (i = 0, k = 0; i < SIZE; i++)
  if (ArrayA[i] % 2 == 0)  // если четное число в массиве ArrayA
  {
    ArrayB[k] = ArrayA[i];
    k++; // увеличиваем индекс второго массива
  }
```

По завершении цикла в массиве **ArrayB** будет **k** элементов.

В функцию, реализующую данный алгоритм, нужно передавать указатели на оба массива и их исходные размеры, а возвращать можно количество скопированных во второй массив элементов.

```
int CopyEven(int *Arr1, int *Arr2, int Size)
{
  int i, k;
  for (i = 0, k = 0; i < Size; i++)
    if (Arr1[i] % 2 == 0)  // если четное число в массиве Arr1
    {
      Arr2[k] = Arr1[i];
      k++;
  }
  return k;
}
```

Пример использования функции:

```
#define SIZE 100
int main()
{
  int ArrayA[SIZE], ArrayB[SIZE];
  int sizeB;
//  ... Заполнение массива ArrayA
  sizeB = CopyEven(ArrayA, ArrayB, SIZE); // Копируем
  ViewArray(ArrayA, SIZE);// Выводим SIZE элементов массива ArrayA
  ViewArray(ArrayB, sizeB); // Выводим sizeB элем-в массива ArrayB
//...
  return 0;
}
```

5. Изменение порядка следования элементов без изменения размеров исходного массива.

Примеры задач данного типа: замена значений некоторых элементов на другие в соответствии с определенными правилами, перестановка и сдвиг элементов различного характера, сортировка массивов по убыванию, возрастанию, и по любому другому признаку и др.

Особенностью этого класса задач является то, что, несмотря на постоянство количества элементов массива, для перестановки или замены элемента сначала

этот элемент необходимо найти. Таким образом, чтобы преобразовать массив, требуется несколько раз его просмотреть, поэтому обычно требуется применение нескольких вложенных циклов.

Пример 10. *Сделать перестановку-инверсию массива*

Перестановка-инверсия – это такая перестановка, когда первый элемент становится последним, второй – предпоследним и т.д. Например, массив, элементы которого равны **1, 2, 3, 4, 5** после перестановки-инверсии должен выглядеть так: **5, 4, 3, 2, 1**.

Пусть размер массива **N**. Тогда элемент **A[0]** надо переставить с **A[N-1]**, **A[1]** с **A[N-2]** и т.д. Но перебор нужно делать только до средины массива, т.к. перестановка затрагивает одновременно и первую, и вторую половину массива. Если в цикле пройти по всем элементам массива, сначала все элементы будут переставлены правильно, а затем (когда **i > N/2**) будут возвращены на свои места.

```
for (i = 0; i < N/2; i ++) // перебор только до середины!
{
  c = A[i];
  A[i] = A[N-1-i];
  A[N-1-i] = c;
}
```

Пример 11. *Сделать циклический сдвиг массива*

При циклическом сдвиге (например, вправо) первый элемент переходит на место второго, второй на место третьего и т.д., а последний элемент – на место первого. Например, массив, элементы которого равны **1, 2, 3, 4, 5** после циклического сдвига должен выглядеть так: **5, 1, 2, 3, 4**.

Для выполнения циклического сдвига нам будет нужна одна временная переменная – в ней мы сохраним значение последнего элемента, пока будем переставлять остальные. Обратите внимание, что мы начинаем с конца массива, иначе массив просто заполнится первым элементом. Первый элемент ставится отдельно – копированием из временной переменной.

```
c = A[N-1];  // запомнить последний элемент
for (i = N-1; i > 0; i--) // цикл с уменьшением i
  A[i] = A[i-1];
A[0] = c;     // первый элемент ставим отдельно
```

Если оформить данный алгоритм в виде функции, ее можно будет использовать для циклического сдвига на **k** элементов просто повторяя в цикле вызов данной функции **k** раз.

Пример 12. *Переформировать массив таким образом, чтобы сначала шли все положительные элементы, затем нулевые, а в конце – отрицательные.*

Данную задачу проще решить, используя дополнительный массив. При этом необходимо учитывать, что номера элементов в исходном и новом массиве не совпадают. В качестве индекса исходного массива мы, как и ранее, будем использовать переменную цикла. А для указания номера элемента в новом массиве нам понадобится специальный индекс (индексы), который придется изменять отдельным оператором.

Задачу можно решить, используя три цикла, в которых в новый массив последовательно переписываются сначала положительные элементы, затем нулевые и наконец – отрицательные. Однако есть более оптимальный вариант: одновременно переписываем в начало нового массива положительные элементы, а в конец – отрицательные. Для этого используем два дополнительных индекса, причем первый будет увеличиваться с начала массива, а второй уменьшаться с конца.

Для того, чтобы в средине оказались нулевые элементы, их не надо специально переписывать, можно просто заранее обнулить все элементы второго массива

```
  #define SIZE 100
  int Array[SIZE], ArrayDop[SIZE];
  int i, j, k;
  // заполняем массив Array
  for (i = 0; i < SIZE; i++)
    ArrayDop[i] = 0; // очищаем второй массив

  for (i = 0, j = 0, k = SIZE -1; i < SIZE; i++)
  {
    if (Array[i] > 0) // если положительное число в массиве Array
    {
      ArrayDop[j] = Array[i]; // Переписываем его с начала массива
      j++;  // индекс идет от начала в концу
    }
    if (Array[i] < 0) // если отрицательное число в массиве Array
    {
      ArrayDop[k] = Array[i]; // Переписываем его с конца массива
      k--;  // индекс идет от конца в началу
    }
  }
```

При создании функции, реализующей данный алгоритм, нет смысла передавать в функцию два массива, вспомогательный массив можно создать прямо в функции, не забывая после перестановки скопировать из него переставленные значения обратно в исходный массив.

```
void ArrRearrange(int *Arr, int Size)
{
  int * ArrayDop;
  int i, j, k;

  ArrayDop = new int[Size]; // создаем динамический массив
  for (i = 0; i < Size; i++)
    ArrayDop[i] = 0; // очищаем этот массив

  for (i = 0, j = 0, k = Size-1; i < Size; i++)
  {
    if (Array[i] > 0) // если положительное число в массиве Array
    {
      ArrayDop[j] = Array[i]; // Переписываем его с начала массива
      j++;  // индекс идет от начала в концу
    }
    if (Array[i] < 0) // если отрицательное число в массиве Array
    {
      ArrayDop[k] = Array[i]; // Переписываем его с конца массива
      k--;  // индекс идет от конца в началу
    }
  }
  for (i = 0; i < Size; i++)
    Array[i] = ArrayDop[i]; // копируем переставленные значения
}
```

Пример использования функции:

```
#define SIZE 100
int main()
{
  int Array[SIZE];
//  ... Заполнение массива Array
  ViewArray(Array, SIZE);     // Вывод исходного массив
  ArrRearrange(Array, SIZE); // Перестановка массива
  ViewArray(ArrayA, SIZE);    // Вывод переставленного массива
//...
  return 0;
}
```

Различные алгоритмы сортировки, а также задачи шестого типа – на переформирование массива с изменением его размеров – будут рассмотрены в последующих работах.

***Пример 13**. Применить разработанные функции в программе*

Для примера напишем программу, в которой, используя созданные функции, можно было бы задавать и изменять размеры массивов, заполнять исходный массив случайными числами или по порядку, копировать из первого массива во второй четный числа, находить минимальные значения в массивах и отображать содержимое массивов на дисплее.

Для улучшения структуры программы создадим простейшее меню.

```
int main(int argc, char* argv[])
{
  int* Array1 = 0;  // Указатель на первый массив
  int* Array2 = 0;  // Указатель на второй массив
  int x1, k, m;
  int Size1 = 10, Size2 = 10; // размеры массивов
  char key;

  srand(time(0));   // Для генератора случайных чисел
  setlocale(0, "Russian");

  Array1 = new int[Size1];  // Выделяем память под массив
  Array2 = new int[Size2];  // Выделяем память под массив
  for (;;) // Бесконечный цикл
  {
    printf("\n  Выберите пункт меню:\n");
    printf("  0 - Выйти из программы\n");
    printf("  1 - Изменить размеры массивов\n");
    printf("  2 - Задать массив 1 случайными числами\n");
    printf("  3 - Задать массив1 по возрастанию\n");
    printf("  4 - Вывести массив 1\n");
    printf("  5 - Вывести массив 2\n");
    printf("  6 - Найти мин. значения массивов\n");
    printf("  7 - Копир-ть из 1-го масс. во 2-ой четные числа\n");
    key = _getch();

    switch (key)
    {
    case '0':
      delete[] Array1;
      delete[] Array2;
      return 0;   // Завершение программы
    case '1':
```

```
      delete[] Array1;
      delete[] Array2;
      printf("Размер массивов = ");
      scanf("%d", &Size1);
      Size2 = Size1;
      Array1 = new int[Size1];  // Выделяем память под массив
      Array2 = new int[Size2];  // Выделяем память под массив
      printf("Массивы созданы\n");
      break;
    case '2':
      FillArrayRand(Array1, Size1);// Заполнение случ.числами
      printf("Массивы заполнены\n");
      break;
    case '3':
      printf("Заполнить от: ");
      scanf("%d", &x1);
      printf("с шагом: ");
      scanf("%d", &k);
      FillArrayStep(Array1, Size1, x1, k);
      printf("Массивы заполнены\n");
      break;
    case '4':
      printf("Массив 1:\n");
      ViewArray(Array1, Size1);
      break;
    case '5':
      printf("Массив 2:\n");
      ViewArray(Array2, Size2);
      break;
    case '6':
      m = FindMin(Array1, Size1);
      printf("Мин. число массива 1: %d\n", m);
      m = FindMin(Array2, Size2);
      printf("Мин. число массива 2: %d\n", m);
      break;
    case '7':
      Size2 = CopyEven(Array1, Array2, Size1); // копируем четные
      printf("Массив 1:\n");
      ViewArray(Array1, Size1);
      printf("Массив 2:\n");
      ViewArray(Array2, Size2);
      break;
    }
  } // Конец цикла for
}
```

Функции заполнения и отображения массивов было рассмотрены в лабораторной работе 10.

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

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

Задание. Написать программу для работы с массивами. Программа должна содержать **меню** и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью **функций**.

Для получения максимального балла добавить в меню программы пункты:
– с возможностью циклического сдвига выбранного массива
– с возможностью переформирования массива таким образом, чтобы сначала шли все четные элементы, а в конце – нечетные.

Варианты заданий

1. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива найти сумму цифр и записать ее в соответствующий элемент второго массива (использовать функцию из задания 1 лаб. работы 10). Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

2. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива найти количество нечетных цифр и записать ее в соответствующий элемент второго массива (использовать функцию из задания 1 лаб. работы 10). Находить средние значения массивов. Выводить массивы и другую информацию на дисплей.

3. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива используя функцию из задания 1 лаб. работы 10 переставить цифры и записать полученное число в соответствующий элемент второго массива. Находить средние значения массивов. Выводить массивы и другую информацию на дисплей.

4. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива используя функцию из задания 1 лаб. работы 10 найти среднее значение цифр составляющих это число и записать полученное среднее в соответствующий элемент второго массива. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

5. Написать программу для работы с тремя целочисленными массивами A, B и C. Программа должна содержать меню и позволять проводить повторные

вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первые два массива двумя способами: случайными числами и по порядку от x1 с шагом k. Используя функцию из задания 1 лаб. работы 10, вычислить среднее арифметическое чисел от a до b включительно, где a берется из A[i], a b – из B[i], для каждого i и записать вычисленное среднее в C[i]. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

6. Написать программу для работы с тремя целочисленными массивами A, B и C. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первые два массива двумя способами: случайными числами и по порядку от x1 с шагом k. Используя функцию из задания 1 лаб. работы 10, вычислить среднее геометрическое чисел от a до b включительно, где a берется из A[i], a b – из B[i], для каждого i и записать вычисленное среднее в C[i]. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

7. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива найти квадратный корень с точностью до 0.0001 (используя функцию из задания 1 лаб. работы 10) и записать его в соответствующий элемент второго массива. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

8. Написать программу для работы с двумя вещественными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива найти дробную часть вещественного числа, округленного до 4 знака после запятой (используя функцию из задания 1 лаб. работы 10) и записать результат в соответствующий элемент второго массива. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

9. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Найти среднее арифметическое положительных значений и отрицательных значений первого массива. Скопировать во второй массив значения, лежащие в диапазоне между ними. Находить максимальное и минимальное значения массивов. Находить средние значения массивов. Выводить массивы и другую информацию на дисплей.

10. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1

с шагом k. Найти количество элементов, лежащих между первым минимальным и последним максимальным элементами первого массива. Скопировать во второй массив значения, лежащие в диапазоне между ними. Находить максимальное и минимальное значения массивов. Находить средние значения массивов. Выводить массивы и другую информацию на дисплей.

11. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива максимальную цифру и записать ее в соответствующий элемент второго массива (использовать функцию из задания 1 лаб. работы 10). Находить средние значения массивов. Выводить массивы и другую информацию на дисплей.

12. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа первого массива минимальную цифру и записать ее в соответствующий элемент второго массива (использовать функцию из задания 1 лаб. работы 10). Находить разность между минимальным и максимальным значениями каждого массива. Выводить массивы и другую информацию на дисплей.

13. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 найти произведение цифр составляющих это число и записать полученное среднее в соответствующий элемент второго массива. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

14. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 определить, есть ли среди цифр числа одинаковые, и, если есть, то скопировать данное число во второй массив. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

15. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 определить, делится ли сумма цифр числа на 3, и, если да, то скопировать данное число во второй массив. Находить максимальное и

минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

16. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 определить, делится ли произведение цифр числа на 7, и, если да, то скопировать данное число во второй массив. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

17. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Найти количество элементов, лежащих между первым кратным 5 и последним кратным 7 элементами первого массива. Скопировать во второй массив значения, лежащие в диапазоне между ними. Находить максимальное и минимальное значения массивов. Находить средние значения массивов. Выводить массивы и другую информацию на дисплей.

18. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: вручную и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 определить, является ли целое число палиндромом, и, если да, то скопировать данное число во второй массив. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

19. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 найти количество четных цифр данного числа и записать его в соответствующий элемент второго массива. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

20. Написать программу для работы с двумя целочисленными массивами. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять под них память динамически. Заполнять первый массив двумя способами: случайными числами и по порядку от x1 с шагом k. Для каждого числа из первого массива используя функцию из задания 1 лаб. работы 10 определить, является ли число простым, и, если да, то скопировать данное число во второй массив. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.