Базовые  структуры данных

**Структура данных (*data structure*)** – это логическая или математическая модель  организации данных для хранения и обработки определенным набором операций.

**Структура данных**– это программная организация данных, обеспечивающая эффективное хранение и обработку  множества однотипных и/или логически связанных данных.

Операции с данными – это алгоритмы, принимающие на вход определенную структуру данных и обрабатывающие их. Основные алгоритмы обработки структур данных реализуют следующий набор операций:

§  обход элементов структуры данных для работы с каждым элементом;

§  включение/добавление/вставка элементов  в структуру данных;

§  исключение/удаление/извлечение элемента структуры данных;

§  поиск элемента, поиск максимального, минимального элемента структуры данных;

§  перестановки/упорядочивание элементов структуры данных по  правилу;

§  проверка на полноту/пустоту структуры данных;

§  очистка структуры данных – удаление всех элементов.

Базовые структуры данных являются структурами данных, на основе которых идет построение абстрактных структур данных. Рассматриваемые алгоритмы (операции) обработки являются основой для построения алгоритмов  обработки структур данных.

Понятие структуры данных и типа данных различны. Понятие типа данных определяет представление данных в памяти компьютера и  набор операций, применимых для данного типа значений вычислителем. Тип данных определяет:

·        *способ хранения данных*указанного типа, выделение памяти в байтах и представление данных в виде двоичных значений;

·        *диапазон допустимых значений*, которые может принимать переменная указанного типа в силу ограниченности байтов памяти;

·        *множество допустимых операций*, которые определены для переменной заданного типа.

*Базовые типы данных* определяются языком программирования и его стандартом. *Абстрактные типы данных* (АТД) описывают программисты, программируя  способ хранения на базовых типах и набор операций (алгоритмов) в виде подпрограмм. С помощью АТД объявляют сложные переменные, их называют *объектами*. В дальнейшем объектами оперируют для решения конкретных задач.

Структуры данных программируются как АТД на основе базовых типов выбранного языка программирования с описанием связей и алгоритмов основных операций. Таким образом, тип данных является *физическим представлением  данных в  компьютере*, а  структура данных – *логическим представлением*для программирования эффективных алгоритмов. Наиболее используемые структуры данных реализованы в языках программирования как типы, например, массив.

Основной характеристикой любой структуры данных является набор операций и сложность (эффективность) каждой операции.  Эффективность алгоритма операции определяет её быстродействие.[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733" \l "_ftn1" \o ")

Различают следующие базовые структуры данных, на которых строятся основные [абстрактные структуры данных](https://educon2.tyuiu.ru/mod/book/view.php?id=558744): ***массив, динамический список, файлы данных******[[2]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733" \l "_ftn2" \o ")***.

## **Учебное пособие - Массивы**

### **1. Базовые структуры данных**

#### **1.1. Массивы**

Базовой элементарной  структурой данных для построения более сложных структур данных является ***массив***. Массивы реализованы в языках программирования как *тип данных*.

**Массив** – это набор данных, где под одним именем (идентификатором переменной) в памяти компьютера находится множество значений *одного типа*.

Значение в массиве называют *элементом массива*. Каждое значение в массиве имеет порядковый  номер, который называют *индексом*. К элементу массива обращаются, указав *имя массива и индекс элемента*. В одномерном массиве к элементу обращаются, указав один индекс, в двумерном – два индекса, в многомерном – соответствующее мерности массива количество индексов. Количество индексов, описывающих положение элемента в массиве, называют *мерностью массива*.

Рис.1 Одномерный массив

В синтаксисе языка С++ объявление массива – это объявление переменной с указанием  в квадратных скобках количества элементов в массиве, *размера массива.*В памяти компьютера выделяется неделимый кусок памяти, размер которого в байтах определяется как размер типа в байтах умножить на количество элементов массива. Например, количество байтов массива на рис.1 определяется как 5\*size(int)= 5\* 4 байта=20 байтов. Именно поэтому размер массива объявляют всегда целочисленной *константой.*Количество квадратных скобок  в объявлении определяет *мерность массива.*

<тип>  имя массива [размер1]…[размерN];

Рис.2 Двумерный массив

Объявление массива:

const int K = 10;              //константа, один из размеров массива

const int M = 5;

     int arr[K], darr[K];          //одномерный массив

     int mas[K][M], Х[4][8];       //двумерный массив

double А[K][K][3];           //трехмерный массив

Индексы массива в С++ нумеруют всегда начиная с 0. Индексы могут быть заданы целочисленной переменной, константой или целочисленным выражением. Таким образом, если размер массива объявлен константой M со значением  5, то индексы будут изменяться в диапазоне  от 0 до 4.

Рассмотрим простейшие алгоритмы работы с мас­сивами, особенности организации циклов при обработке одномерных массивов (векторов) и двумерных массивов (матриц), организация ввода/вывода массивов, а также общие вопросы организации про­грамм для обработки массивов.

Рис. 3 Модель трехмерного массива

*Обработка массивов* организована последовательным просмотром для доступа к каждому элементу – **обходом** элементов массива, это реализуется организацией циклов. При этом ин­дексы часто используются в качестве управляющих переменных циклов. Существует зависимость количества вложенных циклов от мерности (индексов) массива, сколько индексов имеет элемент массива – столько вложенных циклов надо организовать для полного обхода. Если массив одномерный, то для полного обхода организуем один цикл, если двумерный, то два вложенных цикла, если трехмерный – три вложенных цикла и т.д.

Если массив заполнен символами, то одномерный массив символов называют *строкой*, а двумерный – массивом строк или  *текстом*.

Рассматриваемые алгоритмы являются базовыми для построения более сложных алгоритмов обработки массивов.  Алгоритмы оформлены в виде фрагментов программ или функций, описаны в общем виде применительно к статическим[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1364#_ftn1) массивам произвольных размеров. Границы изменения индексов массива описаны как локальные переменные или глобальные именованные константы.

### **азовые структуры данных**

#### **1.2. Алгоритм обхода элементов массива для ввода/вывода, заполнения значениями**

***Алгоритм обхода  элементов массива  для ввода/вывода, заполнения  значениями***

*Ввод массива* реализуется посредством полного или частичного обхода для записи значений элементов массива в память компьютера. Элементы в памяти расположены линейно, независимо от мерности массива. Если это многомерный  массив, то построчно.

Для обхода элементов массива нужно составить алгоритм, обеспечивающий изме­нение индексов и последовательное заполнение элементов массивов значениями. Удобно использовать цикл с параметром  и известным числом повторений, где параметр цикла используется как индекс для перебора элементов, а число повторений совпадает с размером массива или меньше его.

Источником значений могут быть ресурсы ввода стандартного или файлового, генераторы случайных  чисел, результаты вычислений заданных выражений, правила или другие массивы.

**Программа 1.1** Функции ввода и заполнения массива  случайными значениями

//----------- Ввод  одномерного массива-----------------------------

void GetArray(int arr[], int N){

     for (int i = 0; i < N; i++){

          cin >> arr[i];

     }

}

//----------- Заполнение генератором случайных значений-------------

void RndArray(int arr[], const int N, int a, int b){

     for (int i = 0; i < N; i++){

// диапазон случайных  значений от a до b

       arr[i] = a + rand() % (b - a + 1);

     }

}

//------------------------------------------------------------------

Алгоритм  этой функции  в цикле выполняет  N раз ввод или генерацию случайного значения для очередного элемента массива.

Моделью двумерного массива является матрица с заданным количеством строк и столбцов*.*При вводе двумерного массива необходимо организовать двойной цикл: внешний — по номеру строки, внутренний — по номеру столбца*.*Заполнение массива происходит последовательно по строкам. Если мы организуем внешний цикл по столбцам, а внутренний – по строкам, то заполнение будет происходить столбцами.

**Программа 1.2** Функции ввода двумерного массива

//----------- Ввод   матрицы   по   строкам-------------------------

void GetDArray(int mas[][M], int N){

     for (int i = 0; i < N; i++)

          for (int j = 0; j < M; j++)

                cin >> mas[i][j];

}

//-----------Ввод   матрицы   по   столбцам-------------------------

void GetStDArray(int mas[][M], int N){

     for (int j = 0; j < M; j++)

          for (int i = 0; i < N; i++)

                cin >> mas[i][j];

}

//------------Генерация случайных значений--------------------------

void  RndDArray (int mas[][M], int N, int a, int b){

     for (int i = 0; i < N; i++)

          for (int j = 0; j < M; j++)

          mas[i][j] = a + rand() % (b - a + 1);

}

//------------------------------------------------------------------

В Программе 1.2 N — число   строк,   М — число   столбцов   матрицы. При данном синтаксисе программы  N – локальная переменная, а  M должна быть глобальной константой.

Ввод в приведенной программе выполняется N\*M раз и обеспечивает каждый раз ввод одного элемента массива. Порядок ввода элементов определяется порядком изменения индексов  в программе и по строкам будет следующим: mas[0][0], mas[0][1], … mas[0][M-1], mas[1][0], mas[1][1], . . . mas[1][M-1], … mas[N-1][0], mas[N-1][1], … mas[N-1][M-1].

Ввод по строкам, реализованный в приведенной программе является наиболее естественным, так как в памяти компьютера элементы массива располагаются именно по строкам.   В случае необходимости можно организовать заполнение массива по столбцам. Для этого внешний цикл должен быть организован по номеру столбца (j), а внутренний — по номеру строки (i) с соответствующими границами изменения индексов.

Формула генерации случайных значений не изменилась, но обход осуществляется в двумерном массиве двумя циклами и в данном случае по строкам.

Для обхода трехмерного массива необходимо организовать три вложенных цикла для изменения соответствующего индекса  и доступа к элементу массива.

**Программа 1.3**Фрагмент программы обхода  трехмерного массива для  заполнения его элементов значением функции от параметров циклов и вывода на экран результатов.

//------------------------------------------------------------------

…

for (int i = 0; i < N; i++) {

  for (int j = 0; j < M; j++) {

    for (int k = 0; k < 5; k++) {

       A[i][j][k] = (i + 1) \* 100 + (j + 1) \* 10 + (k + 1);

       cout << setw(4) << A[i][j][k];

     }

     cout << endl;

   }

   cout << endl;

}

…

//------------------------------------------------------------------

**Вывод массива** реализуется посредством полного или частичного обхода для чтение значений элементов в памяти и вывода на внешнее устройство.

При выводе массивов необходимо обеспечить наглядность и удобство восприятия полученных результатов. Вывод одномерного массива, как правило, целесообразно осу­ществлять в строку, сопровождая поясняющим текстом.

**Программа 1.4**  Перегруженные функции  вывода  одномерного массива

//---функция вывода массива ----------------------------------------

void OutArr(int  arr[], int N){

 cout << endl;

 for (int i = 0; i < N; i++){

          cout << setw(4) << arr[i] << " ";

     }

}

//-----вывод двух одномерных массивов одного размера в столбцы-------

void OutArr(int  arr[], int darr[], int N){

      cout << endl;

     for (int i = 0; i < N; i++){

          cout << setw(4) << arr[i] << " "<<darr[i];

         cout << endl;

     }

}

//------------------------------------------------------------------

Вывод нового массива начинается с новой строки. Вывод двух или более массивов различных размеров, как пра­вило, осуществляется в строку.

При выводе двух или нескольких одномерных массивов одного размера часто удобно вывести их как расположенные параллельно столбцы/строки.

Двумерные массивы необходимо выводить в привычном виде (по строкам), начиная вывод новой строки массива в новую строку экрана.

**Программа 1.5** Функция   вывода  двумерного массива одновременно на экран и в текстовый файл.

//---функция вывода двумерного массива------------------------------

void PrintDArray(int arr[][M], int N, ofstream\* file){

     for (int i = 0; i < N; i++){

          for (int j = 0; j < M; j++){

                cout << setw(5) << arr[i][j] << ' ';

                file << setw(5) << arr[i][j] << ' ';

          }

          cout << endl;     //переход на новую строку экрана

          file << endl;

     }

}

//------------------------------------------------------------------

#### **1.3. Алгоритмы сдвига элементов массива для реализации вставки и удаления**

***Алгоритмы сдвига элементов массива для реализации вставки и удаления***

Для реализации операций удаления и вставки элементов в одномерном массиве применяют алгоритмы сдвига элементов влево или вправо.

1.     **Cдвиг влево**  массива размером N

Рис 4. Сдвиг влево элементов одномерного массива

Алгоритм  сдвигает весь «хвост» массива, на­чиная с 0 - го элемента, на одну позицию влево, т. е. выполняя операции **arri=arri+1**, где i=0, 1, . . ., N-2. 0-ой элемент при этом теряется.

**for ( i = 0; i<N-1; i++)**

**arr[i]=arr[i+1];**

Если мы хотим потерять(удалить) К-ый элемент, то надо совершить сдвиг влево на него.

**for ( i = К; i<N-1; i++)**

**arr[i]=arr[i+1];**

2.     **Cдвиг вправо** массива аrr размером N

Рис 5. Сдвиг вправо элементов одномерного массива

Алгоритм передвигает «хвост» массива вправо на одну позицию, выполняя операцию **arri+1=arri,** i= N-2, . . ., 0. Перемещение элементов массива нужно начинать с конца. В противном случае весь «хвост» будет заполнен элементом a [0].

**for ( i = N-2; i>=0; i--)**

**arr[i+1]=arr[i];**

Если мы хотим раздвинуть массив, начиная с K-го элемента, то

**for ( i = N-2; i>=K; i--)**

**arr[i+1]=arr[i];**

Сдвиг влево применяется для удаления элементов из массива, сдвиг вправо – для вставки элементов в массив.

*Удаление элемента из массива.*

Требуется удалить К-й эле­мент из массива А размером N. Удалить элемент, расположенный на К-м месте в массиве, можно, сдвинув весь «хвост» массива, на­чиная с (К+1) - го элемента, на одну позицию влево, т. е. выполняя операции ai=ai+1, i=K, K+1, . . ., N-2 (см. программу 1.6).

**Программа 1.6**Применение функциисдвига  влево и удаление элемента с индексом К=2

//----------функция сдвига влево------------------------------------

void left (int arr[], const int n, int k){

   for (int i=k; i<n-1; i++)

    arr[i]=arr[i+1];

  arr[n-1]=0;              // последнему элементу присваиваем 0

 }

 ...

//-----------вызов функции сдвига для удаления элемента-------------

const int N=6;

int  A[N]={-3,0,4,-6,5,7};

left (A, N, 2);            // К=2

**...**

//------------------------------------------------------------------

*Включение элемента в заданную позицию массива.*

Перед включением элемента в К-ю позицию необходимо раздвинуть мас­сив, то есть передвинуть «хвост» массива вправо на одну позицию, выполняя операцию ai+1=ai**,** i=N, N-2, . . ., К. Перемещение элементов массива нужно начинать с конца. В противном случае весь «хвост» будет заполнен элементом a[К]. Далее, К-му элементу присваивается заданное значение  В.  Необходимо предусмотреть, что размер массива увеличивается на 1 (см. программу 1.7).

**Программа 1.7**Применение функциисдвига  вправо и вставка элемента B с индексом К=3

//---функция сдвига вправо------------------------------------------

void right (int arr[], const int n, int k){

   for (int i=n-2; i<=k; i--)

    arr[i+1]=arr[i];

  arr[k]=0;              // k-му элементу присваиваем 0

 }

 ...

//------вызов функции сдвига для вставки элемента-------------------

const int N=7;

int K=3;

int  A[N]={-3,0,4,-6,5,7};

right (A, N-1, K);

a[K]=B;

  ...

//------------------------------------------------------------------

При вставке и удалении необходимо учитывать границы массива. При вставке элементов размер массива должен быть объявлен заведомо большим, чем исходное число элементов (N+P, где Р – число вставляемых элементов).

*Включение элемента в массив, упорядоченный по возраста­нию, с сохранением упорядоченности*

Сначала необходимо найти элемент, перед которым необходимо включать заданное значение В. Для этого нужно проверять условие В>А[i] для i=0, 1, ... При выполнении условия текущее значение индекса (i) определяет позицию нового элемента. Далее, включение элемента осуществ­ляется в соответствии с алгоритмом сдвига вправо.

**Программа 1.8**Фрагмент программы включения элемента в массив.

//------------------------------------------------------------------

...

     k=0;

//--- доходим до места согласно порядку, проверяя границу массива-----

 while (B>a[k] && k<N)   k++;

for ( i = N-1; i>= k;  i--)    a[i+1]=a[i];

     a[k]=B;

...

//------------------------------------------------------------------

*Удаление строки из матрицы*

Требуется удалить строку массива А[N][M] с заданным номером К. Решение задачи аналогично удалению элемента из одномерного массива. Все строки, начиная с (К+1)-й нужно переместить вверх. Число строк уменьшается на 1. Удаление столбца осуществляется аналогично.

**Программа 1.9**Фрагмент программы удаление строки из матрицы.

//------------------------------------------------------------------

…

for ( i = k; i<N; i++)

    for ( j = 0; j<M; j++)

          A[i][j]= A[i+1][j];

L = N-1;                          // новое число строк

**…**

//------------------------------------------------------------------

*Включение строки в матрицу*

Включаемая строка задана как одномерный массив (вектор) С. Включение строки в массив А[N][M]  аналогично включению элемента в одномерный массив.

**Программа 1.10**Фрагмент программы включения строки в матрицу

//------------------------------------------------------------------

…

for ( i = N-1; i>=k; i--)

    for ( j = 0; j<M; j++)

          A[i+1][j]= A[i][j];

for ( j = 0; j<M; j++)

     A[k][j]=C[j];

 …

//------------------------------------------------------------------

Прежде всего, пе­ремещают строки, начиная с К-й, вниз (в обратном порядке). Пере­мещение одной строки связано с пересылкой всех элементов этой строки, что требует организации цикла по номеру столбца. Далее включают строку С в качестве К-й.

Включение столбца осуществляется ана­логично.

#### **1.4. Алгоритмы перестановки и преобразования массивов**

***Алгоритмы перестановки и преобразования массивов***

*Перестановка элементов в одномерном массиве*

Перестановка i-го и j-ro элементов осуществляется с использованием вспомогательной пере­менной (Р), в которую временно помещается один из элементов массива:

**P=A[i]; A[i] = A[j] ; A[j]=P;**

Это типовое действие лучше определить как функцию Swap, Программа 1.11.

*Перестановка строк двумерного массива*

С использованием вспомогательной переменной Р пере­становка осуществляется во всех столбцах двух строк циклом.

**Программа 1.11**Фрагмент программы перестановки строк двумерного массива.

//---перестановка элементов в массиве-------------------------------

void Swap(int& a, int& b){

     int temp = a;

     a = b;

     b = temp;

}

     …

Swap(A[i],A[j]);

     …

//----перестановка i-ой и j-ой строк матрицы------------------------

  for ( k = 0; k < M; k++)

   {P=A[i][k]; A[i] [k]= A[j][k] ; A[j][k]=P;}

…

//------------------------------------------------------------------

*Преобразование матрацы в одномерный массив*

Обработка одномерных массивов осуществляется быстрее, чем двумерных того же размера, что часто требует выполнения указанного преоб­разования.

Требуется переслать элементы матрицы размером N\*M в од­номерный массив того же размера по строкам с сохранением поряд­ка следования элементов. Для этого нужно соответствующим обра­зом согласовать индексы исходного массива А[N][M] и формируемого X.

**Программа 1.12**Фрагмент программы преобразования матрицы в одномерный массив.

//------------------------------------------------------------------

    …

int X[N\*M];         // объявление нового  одномерного массива

 for ( i = 0; i<N; i++)

    for ( j = 0; j<M; j++)

          X[i\*M + j]=A[i][j];

…

//------------------------------------------------------------------

*Упорядочивание (сортировка) одномерного массива*

**Алгоритм сортировки** – это алгоритм  упорядочивания элементов в массиве по заданному правилу. Сортировка  числовых массивов – это перестановка элементов либо по возрастанию, либо по убыванию. Символьные массивы являются по сути тоже числовыми так как, сортируют по алфавиту не символы, а числовые коды символов по возрастанию или убыванию.

Сортировка массива может осуществляться как по значениям, так и по ключу. Ключ сортировки – поле, служащее критерием порядка, в случае, когда элемент списка имеет несколько полей. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Рассмотрим простейшие алгоритмы сортировки:

  Сортировка выбором, линейная сортировка. Основная идея алгоритма: находит в  массиве минимальное(максимальное) число и вставляет его на первое место. Затем повторяет операцию для остальных чисел

  Сортировка вставками. Основная идея алгоритма: на каждом шаге алгоритма выбирается один из элементов и вставляется на нужную позицию в уже отсортированном списке до тех пор, пока набор входных данных не будет исчерпан.

  Сортировка обменом,  пузырьковая сортировка. Основная идея алгоритма: Осуществляются перестановки соседних элементов в порядке возрастания (убывания) до тех пор, пока перестановки не потребуются.

  Быстрая сортировка, улучшенный алгоритм. Основная идея алгоритма: выбрать из массива опорный элемент; сравнить все элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, «меньшие опорного», «равные» и «большие»; для «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

**Программа 1.13**Функция линейной сортировки одномерного массива.

//------------сортировка выбором- линейная-------------------------

void LineSort ( int n, int mass[ ]){

       for (int i = 0 ; i < n-1; i++)

          for (int j = i+1 ; j < n ; j++)

             if (mass[ j ]< mass[ i ]){

                 int temp=mass[i];

                 mass[ i ] = mass[ j ];

                 mass[ j ] = temp;

              }

 }

//------------------------------------------------------------------

**Программа 1.14**Функция сортировки вставками.

//--------------сортировка вставками------------------------------

void Sort(int n, int mass[]){

    int newElement, location;

     for (int i = 1 ; i < n; i++) {

        newElement = mass[i];

        location = i - 1;

        while (location >= 0 && mass[location] > newElement){

            mass[location + 1] = mass[location];

            location = location - 1;

        }

        mass[location + 1] = newElement;

    }

}

//------------------------------------------------------------------

**Программа 1.15**Функцияпузырьковой сортировки.

//---- сортировка обменом - пузырьковая сортировка------------------

void  BubbleSort ( int n, int mass[ ]){

     bool flag=true;

     while (flag)

     {    flag=false;

          for (int  i=0 ; i < n-1 ; i++)

             if (mass[ i ]< mass[ i+1])

              {

                 int temp=mass[ i ];   mass[ i ] = mass[i+1];

                 mass[ i+1 ] = temp;

                 flag= true;

              }

      } }

//------------------------------------------------------------------

Рис 6. Алгоритм быстрой сортировки одномерного массива

**Программа 1.16**Рекурсивная[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1367#_ftn1) функция быстрой сортировки.

//-------------  быстрая сортировка ----------------------------------

void qs(int arrау[], int first, int last)

{

    if (first < last)

    {

   int left = first, right = last,

     m = (left + right) / 2; //значение среднего элемента

        do

        {

            while (arrау [left] < arrау [m]) left++;

            while (arrау [right] > arrау [m]) right--;

            if (left <= right)

            {

                int tmp = arrау [left];

                arrау [left] = arrау[right];

                arrау [right] = tmp;

                left++;

                right--;

            }

        } while (left <= right);

        qs(arrау, first, right);

        qs(arrау, left, last);

    }

}

//------------------------------------------------------------------

#### **1.5. Алгоритмы поиска в массиве**

***Алгоритмы поиска в массиве***

Рассмотрим алгоритмы последовательного (линейного) и двоичного (бинарного) поиска. Это два представителя из большого класса алгоритмов, осуществляющих поисковый процесс.

В массивах эффективно искать не значение элемента, а его индексы. Индекс позволит узнать и место элемента и значение, как результат поиска он содержит больше информации.

*Последовательный (линейный) поиск элемента в массиве***.**

Смысл линейного поиска состоит в последовательном переборе элементов списка и сравнении элементов со значением ключа.

**Программа 1.17**Фрагмент программы линейного поиска элемента в одномерном массиве

//--------функция линейного поиска по значению элемента-----------

 int search(int A[],int n,int key) {

      for (int i=0;i<n;i++)

         if (A[i]==key) return i;

  return -1;

 }

…

const int size=12;

int X[size]={-7, 0, 6, 4, 9, 3, -4, 8, 0, 2, 1,-3};

int index= search(X, size, 8);

…

//------------------------------------------------------------------

 Функция в качестве параметров использует массив, количество элементов, ключ поиска.  Ключом будет искомое значение элемента. Если в программе 1.17 будем искать индекс элемента, значение которого равно -8, то значением ключа key является  8. Как результат функция вернет индекс соответствующего элемента (index=7); при неудачном поиске функция возвращает -1, как несуществующий индекс.

Последовательный поиск применим для любой структуры данных, так как основным методом поиска является алгоритм полного обхода элементов. Если алгоритм обхода реализован, то операция сравнения в процессе обхода  позволит осуществить поиск. Последовательный поиск – это полный обход данных плюс операция сравнения с ключом поиска.

Средняя эффективность[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1368#_ftn1) последовательного поиска составляет O(n).

*Поиск минимального (максимального) элемента в массиве.*

Требуется найти минимальный элемент в массиве и его значение поместить в переменную Р, а индекс — в  переменную min.

Для одномерного массива поиск минимального элемента осу­ществляется аналогично определению минимального элемента в потоке чисел, с той разницей, что в рассматриваемом случае все числа находятся в памяти (в программе — массив mas), и, чтобы перейти к следующему элементу, достаточно изменить индекс на 1.

**Программа 1.18**Фрагмент программы и функция поиска индекса минимального элемента в одномерном массиве.

//------------функция поиска индекса минимального элемента ---------

int minIndex(int mas[],  int n)

{

     int min = 0;

     for (int i = 1; i < n; i++)

          if (mas[min] > mas[i]) min = i;

     return min;

}

…

const int size=12;

int X[size]={-7, 0, 6, 4, 9, 3, -4, 8, 0, 2, 1,-3};

int index= minIndex(X, size);

cout<< “Минимальное значение в массиве”<< X[index]<<endl;

cout<< “Индекс минимального значения в массиве”<< index;

…

//------------------------------------------------------------------

Если в массиве несколько элементов имеют минимальное зна­чение, то в min будет запоминаться индекс первого из них. Если проверять условие mas[min] >= mas[i], то будет запоминаться индекс последнего элемента.

Для поиска максимального элемента нужно изменить знак сравнения в условии проверки (mas[min] < mas[i]).

Реализация алгоритма, когда он вызывает сам себя в качестве вспомогательного, называется ***рекурсией***(от латинского recursio—возвращение).

**Рекурсией  в алгоритмах** можно заменить организацию цикла,  это циклическая система вызовов функции в стеке памяти компьютера. Как следствие, рекурсивная организация алгоритма очень затратна для ресурсов компьютера и влияет отрицательно на быстродействие алгоритма. Там, где нужен эффективный, быстрый алгоритм надо применять циклы.

Тем не менее, модели алгоритмов  рекурсивного вызова позволяют построить очень короткий и понятный программный код и поэтому рекурсивная реализация некоторых алгоритмов  важна.

Рекурсивная реализация поиска минимального элемента в массиве представлена двумя функциями.

**Программа 1.19**Рекурсивная реализация поиска минимального элемента  и его индекса  в одномерном массиве mas.

// ----------------- поиск значения минимального элемента------------

int min(int a, int b) {

    if(a > b)

        return b;

    else return a;

}

int minimum(int mas[], int n) {

    if(n == 2)

        return min(mas[0], mas[1]);

    return min(mas[n-1], minimum(mas,n-1));

}

//---------поиск индекса минимального элемента в массиве-------------

int ind\_min(int posl, int perv, int mas[]) {

    if(mas[perv] > mas[posl])

        return posl;

    else return perv;

}

int index\_min(int mas[], int n) {

    if(n == 2)

        return ind\_min(n-1, 0, mas);

    else return ind\_min(n-1, index\_min(mas, n-1), mas);

}

…

const int size=12;

int X[size]={-7, 0, 6, 4, 9, 3, -4, 8, 0, 2, 1,-3};

cout<< “Минимальное значение в массиве”<<minimum(X,size)<<endl;

      cout<< “Индекс минимального значения”<< index\_min(X, size);

…

//------------------------------------------------------------------

Для двумерного массива алгоритм поиска аналогичный, но нужно для каждой строки просматривать элементы всех столбцов (что требует организации двойного цикла) и запоминать два индекса: номер строки  и номер столбца.

**Программа 1.20**Функция поиска в двумерном массиве.Функция-предикат на входе функции DFind  задает выражение условия  поиска в массиве. Проверке подвергается каждый элемент. Результат поиска значение типа структуры DPair.

//------------------------------------------------------------------

struct DPair{

 int first;

 int second;

};

  DPair  DFind(int arr[][M], int N, bool(\*Predicate)( int)) {

     for (int i = 0; i < N; i++)

          for (int j = 0; j < M; j++)

                if (Predicate(arr[i][j]))      return { i, j };

     return { -1, -1 };

   }

 bool comp( int b) { return !(b % 2); } //проверка на четность

…

 int y[4][M]; RndDArray(y, 4, -4, 5);

      PrintDArray(y, 4);

      DPair temp = DFind(y, 4, comp);

      cout << endl << y[temp.first][temp.second];

//------------------------------------------------------------------

**Программа 1.21**Фрагмент поиска  минимального в двумерном массиве

//------------------------------------------------------------------

*…*

     int indexi, indexj, val = 0;

     for (int i = 0; i < a; i++){

         for (int j = 0; j < b; j++)

         {

             if (val < а[i][j])

             {

                indexi = i;

                indexj = j;

                val = a[i][j];

             }

         }

     }

    …

//------------------------------------------------------------------

**Программа 1.22**Фрагмент поиска в двумерном массиве максимального  элемента каждой строки

//------------------------------------------------------------------

//n - количество строк,

//max [n] - массив, с max элементом каждой строки

 …

 for (int i = 0; i < n;  i++) {

    max[i]=a[i][0];

         for (int j = 0; j < m; j++)       if (a [i][j] > max[i])   max [i] = a [i][j];

}

 …

//------------------------------------------------------------------

*Двоичный (бинарный) поиск элемента в массиве*

Бинарный поиск может использоваться только для упорядоченного списка.

Пусть имеем массив из n элементов.

Индексы в концах списка: low=0, high=n-1.

Алгоритм бинарного поиска:

1. Вычислить индекс серединного элемента массива     mid=(low+high)/2.

2. Сравнить значение в серединном элементе с ключом поиска key.

Если совпадение найдено, возвратить индекс mid для нахождения ключа.

Если A[mid]<key, то проводить повторный поиск в правой половине рассматриваемого списка.

Если A[mid]>key, то проводить повторный поиск в левой половине рассматриваемого списка.

1.  Если искомый элемент не находится в списке, то возвратить индикатор сбоя.

Пример представлен на  рис. 7.  Дан массив целых чисел А. Найти элемент с заданным ключом 33. Программы 1.23 и 1.24.

Рис. 7 Алгоритм двоичного(бинарного) поиска.

**Программа 1.23**Итеративный[[2]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1368#_ftn2) алгоритм бинарного поиска

//------------------------------------------------------------------

int binarysearch(int key, int A[], int n){

    int low, high, mid;

    low = 0;

    high = n - 1;

    while (low <= high)

    {

        mid = (low + high) / 2;

        if (key < A[mid])

            high = mid - 1;

        else if (key > A[mid])

            low = mid + 1;

        else

            return mid;

    }

    return -1;

}

//------------------------------------------------------------------

**Программа 1.24**Рекурсивный алгоритм бинарного поиска

//------------------------------------------------------------------

int bisearch(int key, int A[], int low, int high){

    int mid = (low + high)/2;

    if (high < low)

       return -1;

    if (A[mid] > key)

        bisearch(key, A, low, mid-1);

    else

        if (A[mid] < key)

             bisearch(key, A, mid+1, high);

    else

       return mid;

 }

//------------------------------------------------------------------

При последовательном поиске требуется 8 сравнений. При бинарном поиске – только три. Если  n – размер массива, то количество сравнений составит log2n. Таким образом, средняя эффективность[[3]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1368" \l "_ftn3" \o ") бинарного поиска составляет O(log2n).

#### **1.6. Вычислительные алгоритмы с элементами массива**

***Вычислительные алгоритмы  с элементами массива***

*Сумма элементов массива*

Для одномерного массива  *A={a0, .... аn-1}*необходимо вычислить

**Программа 1.25** Фрагмент вычисление суммы элементов одномерного массива

…

int S=0;

 for ( i = 0; i<N; i++)       S=S+a[i] ;

          cout<< “ Сумма элементов в массиве” << S;

…

Для двумерного массива B размером N\*M необходимо вычислить

**Программа 1.26** Фрагмент вычисление суммы элементов двумерного массива.

//------------------------------------------------------------------

…

int S=0;

 for ( i = 0; i<N; i++)

   for ( j = 0; j<M; j++)

           S=S+b[i][j] ;

cout<< “ Сумма элементов в массиве” << S;

…

//------------------------------------------------------------------

*Суммирование диагональных элементов матрицы, след матрицы*

Для матрицы В размером N\*N необходимо вычислить

**Программа 1.27** Фрагмент вычисление суммы элементов главной диагонали двумерного массива

//------------------------------------------------------------------

…

int S=0;

 for ( i = 0; i<N; i++)

      S=S+b[i][i] ;

cout<< “След матрицы” << S;

…

//------------------------------------------------------------------

*Алгоритмы обхода диагональных элементов матрицы* представлены на рис 8.

            Рис. 8  Фрагменты программ  с алгоритмами обхода  диагональных элементов матрицы

*Суммирование двух массивов*

Для одномерных массивов А и В размером N необходимо вычислить *сi=ai+bi, i= 1,2.... N.*

**Программа 1.28** Фрагмент вычисление суммы элементов двух одномерных массивов

…

int C[N];

for ( i = 0; i<N; i++)

      C[i]=b[i]+a[i];

          …

Для двумерных массивов А и В размером NXM необходимо вы­числить *сi j=ai j+bi j*, *i=l, 2, ..., N; j=1, 2,... М*.

**Программа 1.29** Фрагмент вычисление суммы элементов двух двумерных массивов

…

int C[N][M];

 for ( i = 0; i<N; i++)

   for ( j = 0; j<M; j++)

      C[i][j]=b[i][j]+a[i][j] ;

…

*Суммирование элементов заданной строки/столбца матрицы.*

Для  матрицы В размером N\*M необходимо вычислить

т.е. необходимо вычислить сумму элементов i-ой строки, а значит надо изменять индекс столбцов по j.

**Программа 1.30** Фрагмент суммирования элементов заданной строки матрицы.

 …

i=5;

int S=0;

for ( j = 0; j<N; j++)   S=S+b[i][j] ;

 …

Необходимо вы­числить сумму элементов *каждой строки* матрицы В размером N\*M. Результат получим в виде вектора D.

**Программа 1.31** Фрагмент суммирования элементов каждой строки матрицы.

…

int D[N]={0};   //обнулим массив

 for ( i = 0; i<N; i++)

    for ( j = 0; j<M; j++)

          D[i]=D[i] + b[i][j] ;

  …

*Транспонирование матрицы.*

Необходимо заменить строки матрицы A размером N\*M ее столбцами, а столбцы — строками (см. рис 9).   Используем для этого  вспомогательный двумерный массив B.

Рис.9  Транспонирование матрицы

**Программа 1.32** Фрагмент транспонирования матрицы в дополнительном массиве

…

int B[N][M];

 for ( i = 0; i<N; i++)

    for ( j = 0; j<M; j++)  B[i][j]=A[j][i] ;

…

Транспонированную матрицу можно получить в исходном мас­сиве А. Для квадратной[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1369#_ftn1) матрицы размером N\*N для этого необ­ходимо поменять местами каждый элемент верхнего треугольника с соответствующим элементом нижнего (диагональные элементы переставлять не нужно). При этом для каждой строки нужно вы­полнять перестановку для элементов, расположенных правее глав­ной диагонали, с элементами соответствующего столбца, располо­женными ниже главной диагонали. При перестановке используем вспомогательную переменную Р, помещая в нее для временного хранения один из переставляемых элементов, чтобы не потерять его значение (см. программу 1.33).

**Программа 1.33** Фрагмент транспонирования матрицы в исходном массиве

**…**

int P;

 for ( i = 0; i<N-1; i++)

    for ( j = 0; j<N; j++)

          { P = A[i][j];    A[i][j]=A[j][i] ; A[j][i] =P ;}

**…**

*Умножение матрицы на вектор*

Вычисление произведе­ния С матрицы А размером N\*M на вектор В размером М.

**Программа 1.34** Фрагмент вычисления произведения матрицы на вектор.

**…**

int C[N]={0};

 for ( i = 0; i<N; i++)

    for ( j = 0; j<M; j++)

          C[i]=C[i]+A[i][j]\*B[j];

**…**

*Умножение матрицы на матрицу*

Умножить мат­рицу А размером N\*K на матрицу В размером К\*М.

**Программа 1.35** Фрагмент вычисления произведения матрицы на матрицу.

**…**

int C[N][M]={0};

 for ( i = 0; i<N; i++)

    for ( j = 0; j<M; j++)

        for (p=0; p<K; p++)

          C[i][j]=C[i][j]+A[i][p]\*B[p][j];

**…**

*Преобразование матрицы путем умножения (деление) строки на число*

1) Умножение (деление) строки на число. Требуется умножить (разделить) все элементы строки (в программе — i-й) на одно и то же число (В), в частности, на какой-либо элемент этой же строки.

**Программа 1.36** Фрагмент  умножения элементов строки на число.

…

for ( j = 0; j<M; j++)

   A[i][j]= A[i] [j]\*B;

…

2) Сложение строк. Требуется к элементам К-й строки приба­вить элементы L-й строки, умноженные на число (В) (см. програм­му 4.19).

**Программа 1.37** Фрагмент сложения строк матрицы.

…

// сложение k-ой и s-ой строк матрицы

for ( j = 0; j<M; j++)

   A[k][j]= A[k] [j]+ A[s][j] ;

…

#### **1.7. Примеры решения задач обработки массивов**

### **Примеры решения задач обработки массивов**

Рассмотрим примеры решения задач с массивами, применяя для организации структуры программы функциональную декомпозицию. Библиотеки функций  оформляют в  отдельные модули и подключают к проекту решения по мере необходимости.

**Задача 1.**Удалить из массива, в котором все элементы различны, максимальный элемент.

После поиска максимального элемента массив "уплотнить", сдвинув все следующие за ним элементы влево. Последнему (самому правому) элементу массива присвоить 0.

*Решение*

Для того, чтобы решить данную задачу, необходимо:

• найти номер максимального элемента k;

• сдвинуть все элементы начиная с k-го на один элемент влево;

• последнему элементу присвоить значение 0.

Рассмотрим решение задачи на конкретном примере. Пусть дан одномерный массив, состоящий из 10 элементов: 6, 3, 4, 7, 11, 2, 43, 8, 1, 5. Номер максимального элемента равен 6 (k=6), то есть начиная с 6-го элемента будем сдвигать элементы на один влево: 6-му элементу присвоим значение 7-го, 7-му присвоим значение 8-го, а 8-му присвоим значение 9-го, на этом сдвиг заканчивается. Таким образом, сдвиг начинается с k-го элемента и заканчивается (n-1)-м (где n — количество элементов в массиве). После этого последнему элементу присвоим 0, а тогда массив примет вид: 6, 3, 4, 7, 11, 2, 8, 1, 5, 0.

Составим программу для удаления[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1370#_ftn1) максимального элемента из одномерного массива, в ней воспользуемся двумя функциями инициализации массива и вывода его на печать. Первая заполняет массив случайными числами, а вторая выводит на печать этот массив. Чтобы последний 0 не выводился на экран, мы  будем  передавать в функцию вывода не только массив, но и количество элементов, которые надо вывести, начиная с 0-го.

Программа 1.38 Программа решения задачи 1.

#include <iostream>

#include <stdlib.h>

#include <time.h>

//-------функция заполнения массива случайными значениями-----------

void rnd(int mas[], const int n, int a, int b)  {

   for (int i=0;i<n;i++)

   mas[i]= a + rand() % (b-a+1);

  }

//---функция вывода массива ----------------------------------------

void put(int mas[], const int n) {

 cout<<endl;

    for (int i=0; i<n; i++)

    cout << mas[i] <<" ";

  cout<<endl;

 }

//--функция сдвига влево элементов массива--------------------------

void left (int mas[], const int n, int k){

   for (int i=k; i<n-1; i++)

    mas[i]=mas[i+1];

  mas[n-1]=0;

 }

//-----------функция поиска индекса максимального элемента ---------

  int maxIndex (const int mas[], const int n){

int max=0;

 for(int  i = 1 ; i < n ; i++)

   if (mas[i]> mas[max]) max=i;

return max;

}

int main(int argc, char\* argv[]){

 const int n=10;

 int A[n], k;       //k – номер максимального элемента

 rnd(A, n, 0, 9);      // заполнение массива случайными значениями

 cout<<endl<<"Исходный массив";

 put(A, n);  // вывод массива

 k= maxIndex(A, n);//вызов функции поиска номера макс. элемента

 left (A, n, k);                // удаление макс. элемента

 cout<<endl<<"Результирующий массив";  put(A, n-1);

 system ("pause");

 return 0;

}

//----------------------------------------------------------

**Задача 2.**Решить предыдущую задачу, считая, что максимальный элемент может встречаться несколько раз.

*Решение*

Когда необходимо удалять несколько элементов, то это лучше всего делать с конца массива, так как иначе нужно будет снова возвращаться к элементу, который только что удаляли (эта проблема возникает в том случае, когда подряд идут два максимальных элемента: если первый удалить, то на его место снова встанет максимальный элемент). Просматривать массив с конца можно при помощи цикла с параметром, который имеет следующий вид: **for ( i=n-1; i>=0; i--) <тело цикла>;** Значение переменной i будет уменьшаться на единицу начиная  от n-1 до 0. Номер максимального элемента позволит узнать значение и сохранить в отдельной переменной **max** , затем просмотрим массив с конца, и если элемент имеет максимальное значение (равен  max), то удалим его, при этом значение счетчика **k**будем увеличивать на 1. Используя все тот же набор функций, напишем  программу.

**Программа 1.39**Программа решения задачи 2.

//------------------------------------------------------------------

int main(int argc, char\* argv[]){

 const int n=10;

 int A[n], k, m;

//k – номер максимального элемента,

//m-счетчик удаляемых элементов

  rnd(A, n, 0, 100);

// вывод массива

 cout<<endl<<"Исходный массив";  put(A, n);

//вызов функции поиска номера макс. Элемента

 k=maxIndex(A, n);

 int max= A[k];   k=0 ;

// удаление всех макс. элементов

 for (int  i=n-1 ; i>=0; i--)

   if (A[i]== max)   {left (A, n, i);    m++;}

 cout<<endl<<"Результирующий массив";  put(A, n-m);

 system ("pause");

 return 0;

}

//------------------------------------------------------------------

**Задача 3.**Вставить число 50  после пятого элемента массива.

*Решение*

Пусть k — номер элемента, после которого мы должны вставить элемент х (k и х будем вводить с клавиатуры). Вставка осуществляется следующим образом:

• первые k элементов массива остаются без изменений;

• все элементы, начиная с (k+1) -го, необходимо сдвинуть вправо;

• элементу с номером (k+1) присваиваем значение х.

Рассмотрим конкретный пример. Пусть дан следующий одномерный массив из N (N=10) элементов:3, -12, 5, 14, 27, -6, 1, -34, 10, -15. Надо вставить элемент со значением 50 после пятого (4-го) элемента массива. Мы получим следующий массив: 3,-12, 5, 14, 27, 50, -6, 1, -34, 10, -15. Таким образом, после вставки в массиве станет 11 элементов, и это надо учесть при описании массива. Будем выводить массив два раза — до и после вставки нового элемента, используя уже известную функцию put(). Составим теперь основную программу с использованием новой right() , которой передаются: k — номер элемента, место которого надо освободить (т.е. на это место вставим новый элемент) и массив. Сдвиг элементов будем начинать с конца массива.

**Программа 1.40**Программа решения задачи 3.

**//---------** функция сдвига вправо  right**----------------------------**

void right (int mas[], const int n, int k){

for (int i = n-1; i>=k; i--)

     mas[i+1]=mas[i] ;

 }

**//----------** основная программа решения**-------------------------**

int main(int argc, char\* argv[])

{

 const int n=11;

 int A[n], k;

 rnd(A, n-1, 0, 10);

// вывод массива

 cout<<endl<<"Исходный массив";  put(A, n-1);

 cout<<endl<<"Номер элемента, после которого вставлять";

 cin>>k;

 right (A, n-1, k);   A[k]=50;

 cout<<endl<<"Результирующий массив";  put(A, n);

 system ("pause");

 return 0;

}

**//------------------------------------------------------------**

**Задача 4.**Вставить данное число после всех элементов массива, кратных 3.

*Решение*

Первое, на что необходимо обратить внимание, — это описание массива: на сколько элементов может увеличиться массив? Максимальное количество элементов, после которых будет вставлен новый элемент, совпадает с количеством элементов массива, ведь может случиться так, что все элементы массива отвечают заданному свойству. Поэтому массив может увеличиться максимум в два раза, а значит, соответствующее описание массива будет следующим:

**const int n=10;**

**int A[2\*n];**

Если мы будем просматривать элементы массива с начала и вставлять новый элемент после элемента с заданным свойством, то номер последнего элемента каждый раз будет меняться, кроме того, придется пропускать ("перепрыгивать") новый (вставленный) элемент, поэтому решение будет не очень эффективным. Удобнее просматривать массив с конца, тогда вставляемый элемент мешать не будет. Составим программу, используя уже ранее описанные функции.

**Программа 1.41**Программа решения задачи 4.

**//----------** основная программа решения**-------------------------**

int main(int argc, char\* argv[]){

 const int n=10;

 int A[2\*n], k=0;

 rnd(A, n, 0, 9);

 cout<<endl<<"Исходный массив";  put(A, n);

 for (int  i=n-1 ; i>=0; i--)

  if (A[i]%3==0)

      {right (A, n+k, i+1); A[i+1]=50;   k++;}

 cout<<endl<<"Результирующий массив";  put(A, n+k);

 system ("pause");

 return 0;

}

**//------------------------------------------------------------**

**Задача  5.** Вычислить произведение положительных, отрицательных элементов и элементов кратных 2 в заданном одномерном массиве.

Унифицируем  алгоритм вычисления произведения для разных категорий величин путем создания и применения функций – предикатов и функций как аргумента другой функции.

*Решение*

Разработаем функцию multy для вычисления произведения элементов массива. Аргументом этой функции будет указатель на функцию, которая задает условие отбора множителей из элементов массива.

Разработаем функции-предикаты, вычисляющие логическое выражение для возврата истины или ложности в зависимости от соответствия величины некоторому условию: положительная, отрицательная, кратная 2.

**Программа 1.42**Программа решения задачи 5.

//--- Функция, где аргумент - указатель на функцию -----------------

int  multy ( int\* mas, int size, bool  (\*fun) (int) ) {

 int  p=1;

  for (int i=0; i< size; i++)

      if ((\*fun)(mas[i])) p\*=mas[i];

  if (p==1) p=0;

  return p;

 }

//------------ функциии-предикаты-----------------------------------

bool pol (int k) {return k>0;}   //положительное

bool otr (int k) {return k<0;}   //отрицательное

bool mod2 (int k) {return k%2==0;}   //кратное 2

//--------- основная программа решения------------------------------

int main(int argc, char\* argv[ ]){

  int m[5]={-1,2,4,-7,0};

  cout<<multy(m, 5, pol);      //применение предиката pol

  cout<<multy(m, 5, otr);

  cout<<multy(m, 5, mod2);

 return 0;

    }

**//------------------------------------------------------------**

**Задача 6.**  Массив студентов. Известны фамилии и оценки в баллах по 5 дисциплинам каждого из 25 студентов группы. Требуется разработать:

-       функции ввода, вывода  студентов;

-       вычисление среднего балла каждого студента;

-       функцию поиска по ФИО студента и вывод записи о нем;

-       функцию поиска студента с максимальным баллом по заданному предмету;

-       функции сортировки массива студентов по ФИО и среднему баллу;

-       функции чтения/записи массива в текстовый файл, то есть хранение данных на внешнем носителе.

*Решение*

Необходимо реализовать решение задачи на основе массива структур (записей) функциональной декомпозицией.

·        Объявляем тип данных элемента массива – это структура как набор данных разного типа:

**struct Student {**

**char FIO[30];**

**int Ball[5];**

**float SrBall;**

**};**

·        Объявляем статический одномерный массив  с типом данных Student:

**Student G[N];**

·        Описываем функции обработки одномерного массива  со структурированным типом данных элементов.  Алгоритмы будут типовыми операциями над одномерным массивом, но элемент массива нужно обрабатывать, учитывая его структурный тип, то есть по полям структуры. Для переменных структурного типа определена операция присваивания, то есть:

**Student A, B;**

**cout<<"FIO"<<endl; cin.get(A.FIO,30);**

**A.SrBall=0;**

**for (int i=0; i<5; i++) {**

**cin>>A.Ball[i]; A.SrBall+=A.Ball[i];}**

**A.SrBall/=5;**

**B = A;**

**Программа 1.43**Программа частичного[[2]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1370#_ftn2) решения задачи 6 «Массив студентов» на основе функциональной декомпозиции.

//----описание типа struct для одного элемента массива--------------------

struct Student {

      char FIO[30];

      int Ball[5];

      float SrBall;

};

const int N=25;  // размер массива - группа

//-----описание функции вывода на экран массива типа Student--------------

void showStudent(Student St[], int size){

     cout<<endl<<"List Studens"<<endl;

     for (int k=0; k<size; k++ ){

            cout<<endl<<St[k].FIO<<' '<<St[k].SrBall;

            for (int i=0; i<5; i++)  cout<<' '<<St[k].Ball[i];

      }

}

//------описание функции добавления в массив   элемента типа Student -----

void addStudent(Student St[], int size){

  char str[30];

  for (int k=0; k<size; k++ ){

        cout<<"FIO"<<endl;cin.get(); cin.get(St[k].FIO,30);cin.get();

        St[k].SrBall=0;

         cout<<"Ball"<<endl;

          for (int i=0; i<5; i++) {

                   cin>>St[k].Ball[i]; St[k].SrBall+=St[k].Ball[i];}

        St[k].SrBall/=5;

      }

}

//--------сортировка массива студентов по среднему баллу------------------

void sortSrBall(Student St[], int size) {

      Student tmp;

      for (int i = 0; i < size - 1; i++)

            for (int j = i + 1; j < size; j++)

                  if (St[i].Srball < St[j].Srball){

                        tmp = St[i]; St[i] = St[j]; St[j]=tmp;

                  }

}

//------------------------------------------------------------------------

int main(){

//-- объявление массива структур

       Student G[N];

       int count=0; // количество  студентов в группе

       cout<<"count "; cin>>count;

//--вызов функций обработки  массива G типа Student

      addStudent(G,count);

      showStudent(G,count);

      sortSrBall(G, count);

      showStudent(G,count);

      cout<<endl;

      system("pause");

      return 0;

}

//------------------------------------------------------------------------

#### **2.1. Понятие о типе «указатель»**

**Понятие о типе «указатель»**

**Указатель** – это тип данных, который предназначен для хранения физических адресов других переменных. Переменная-указатель хранит адрес[[1]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1372#_ftn1) области памяти, которую занимает другая переменная, тем самым указывая на нее.

*Основное назначение указателей*состоит в том, чтобы обеспечить механизм использования в программе **динамических переменных**.

В  языках программирования различают виды указателей[[2]](https://educon2.tyuiu.ru/mod/book/view.php?id=558733&chapterid=1372#_ftn2):

§  *типизированные указатели -*содержат адрес  памяти определенного типа:

*<тип данных> \*<имя указателя>;*

§  *нетипизированные* указатели *-* содержат адрес  памяти, не связанный с данными какого-либо определенного типа и его перед использованием надо приводить к требуемому типу:

*void  \*<имя указателя>;*

Таблица 1.

Операции с указателями в C++

|  |  |  |
| --- | --- | --- |
| **Операция** | **Действие** | **Пример** |
| \* | обращение к значению, доступ к ячейке памяти, адрес которой хранит указатель, операция разыменования:  <имя переменной> = \*<имя указателя>;  \*<имя указателя> = <значение>; | \*ptr = 25; |
| & | получить физический адрес переменной, значения для указателя:  <имя указателя> = &<имя переменной>; | ptr = &A;    см. рис 11 |
| == | сравнивать на равенство | ptr == ptr1 |
| != | сравнивать на неравенство | ptr != NULL |
| NULL | константа, нулевой указатель,  адреса нет в переменной; нулевой указатель может быть задан числом 0 | p =**NULL**;  p=0; |
| new | операция выделения памяти в куче, возвращает адрес выделенной памяти | int\* ptr;  ptr = new int;  ...  delete ptr; |
| delete | операция освобождения памяти в куче, указателю будет присвоено значение NULL. |

Рис. 11  Обращение к переменной через указатель.