**Контрольні питання**

1. Массивы могут иметь любое число измерений. Однако на практике чаще всего используются ***одномерные*** и ***двумерные*** массивы. Хорошим примером двумерного массива служит шахматная доска или матрица чисел. Двумерные массивы также можно инициализировать (для большей наглядности удобно сгруппировать данные построчно, заключив их в фигурные скобки). Для обхода двумерных массивов используются ***вложенные*** циклы **for**.   
   Т. к. в ОЗУ компьютера данные двумерных массивов расположены последовательно по строкам, то можно применить последовательный обход его элементов с помощью арифметики указателей.
2. 35
3. Массивы при объявлении можно ***инициализировать***, поставив после имени знак **=** и затем в ***фигурных*** скобках записать список значений, разделенных ***запятыми***.   
    **double dArray[5]={1.5, -0.6, 2.5e-3, 3., 1};** При инициализации можно опустить размер массива – будет создан массив, достаточный для хранения всех инициализирующих значений.  
    **double dArray[]={1.5, -0.6, 2.5e-3, 3., 1};**   
   Размер такого массива можно вычислить, используя **sizeof** :  
    **const unsigned int dArraySize=sizeof(dArray)/sizeof(dArray[0]);**

**Нельзя инициализировать больше элементов, чем объявлено в массиве.** Меньше – можно, неинициализированные элементы будут иметь неопределенные значения.

1. Для хранения данных массивов всегда выделяется *непрерывная* область оперативной памяти. Поэтому обход массива можно выполнить, применив *арифметику указателей* (*адресную арифметику*) вместо индексов. Адресная арифметика позволяет выполнять инкрементирование (**++**), декрементирование (**- -**), сравнение **(= =, > , < , >=, <=, !=**) и присваивание (**=**) указателей. Причем операторы ++ и **- -** изменяют значение указателя не на арифметическую единицу, а на адресную единицу (**char** – 1 б.; **int** – 2 б.; **long** – 4б.;   
   CAT – **sizeof**(CAT) б.). Арифметика указателей часто применяется в системном программировании для повышения эффективности программы.
2. Удобно написать функции для выполнения этих алгоритмов, передавая в них параметром массив и его размер. Нужно помнить, что ***массивы*** передаются в функции всегда ***по ссылке*** и функция работает с оригиналом массива, а не его копией. В библиотеке функций языка **С** есть функции с именами **qsort(), lsearch()** и **bsearch()** для выполнения быстрой сортировки, линейного и двоичного поиска в массивах с данными встроенных типов. Стандартная библиотека шаблонов контейнеров **С++** (**STL**) также имеет в своем составе средства для выполнения сортировки и поиска как данных встроенных типов, так и объектов классов.
3. В массиве можно хранить объекты как ***встроенных*** типов данных, так и ***пользовательских***, создаваемых на основании классов. *Класс должен иметь конструктор по умолчанию без аргументов*, и объекты создаются уже при объявлении массива. Доступ к членам-данным массива объектов представляет собой двухступенчатый процесс: вы идентифицируете элемент массива, пользуясь ***оператором индексации*** ( **[ ]** ), а затем добавляете ***оператор доступа*** к члену класса или вызову функции класса (операторы (**.**) или (**🡪** ) ).
4. Массивы размещают свои элементы в ***стековой памяти***. Область памяти стека довольно ограничена, тогда как ***свободной*** памяти намного больше. Существует возможность создания ***массива указателей*** в стеке, размещения объектов в свободной памяти и сохранения в массиве указателей адресов этих объектов. Массивы указателей снижают количество стековой памяти программы.  
    **Cat \*Family[500]; // массив указателей на объекты типа Cat**
5. Можно разместить и ***весь*** ***массив*** в свободной памяти, называемую также кучей (**heap**), сохранив в ***стеке*** только ***указатель*** на область свободной памяти, применив оператор **new** совместно с оператором индексации.