**Контрольні питання**

1. ***Массив*** – совокупность ячеек для хранения данных одного и того же типа. Каждая ячейка массива называется ***элементом*** массива
2. **Индекс массива** — целое число, которое указывает на конкретный элемент **массива**. **Индекс массива** — это не номер элемента, а смещение
3. *Для объявления массива сначала пишется тип, затем имя массива и максимальный размер.* **Р*азмер*** – это число элементов массива, заключенное в квадратные скобки. Например : **long lArray[6];.** Когда компилятор встречает такое объявление, то он резервирует память, достаточную для хранения всех **6** элементов типа **long**, т.е **sizeof(long) \* 6 = 24 б**.
4. *Первый* – 0, последний – 24
5. При выполнении действий с определенным элементом массива компилятор просто вычисляет его местоположение относительно начального элемента массива путем умножения смещения на размер элемента массива и записывает данные по вычисленному адресу, не проверяя при этом принадлежит ли эта область памяти массиву или нет. Подобная ошибка получила название “***запись данных за границы массива***” (**boundary error**) . *Ответственность за нее несет программист.*
6. 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.
7. 1000
8. 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.
6. **int N=1000;**  // **это уже переменная, а не константа!**  
    **Cat \*Family=new Cat[N]; // массив в свободной памяти**Указатель **Family** содержит адрес начального элемента массива. Доступ к каждому элементу такого массива возможен с как помощью индексов , так и применением *арифметики указателей*:  
    **Cat \*pCat=Family; // указатель для перемещения по массиву  
    pCat++; // переход к следующему элементу массива  
    pCat--; // переход к предыдущему элементу массива**Для освобождения свободной памяти, занимаемой массивом нужно использовать оператор **delete** совместно с оператором индексации.  
    **delete [ ] Family;**
7. С*трока символов С-стиля – массив элементов типа char, последний элемент которого содержит нуль-символ* (**null character**) – символ с номером **0** в таблице символов.   
   Строку можно объявить, инициализировать и обработать как и любой другой массив, например : **char str[12]={‘H’,’e’,’l’,’l’,’o’, ’ ‘, ’w’,'o','r','l','d', '\0'};**

**Null-символ (‘\0’), имеет очень важный смысл – это признак конца строки С-стиля.   
Он необходим для правильной работы встроенных строковых функций С/С++.**

1. **cin.get()**
2. определение длины строки – **strlen();  
   char str[12]=”Hello world”; // Размер строки определил программист, учитывая '\0'  
    char str[] =”Hello world”; // Размер строки определил компилятор и добавил ‘\0’**
3. - копирование (присваивание) строк – **strcpy(), strncpy();**
4. - соединение строк – **strcat(), strncat();**
5. - сравнение строк – **strcmp(), strncmp()**
6. Современные С++ компиляторы поставляются с библиотекой классов, включающей большой набор классов для обработки данных. В компиляторе **Borland C++ 5.02** есть специальный класс для работы со строками. Задача строкового класса – преодолеть ограничения обычных символьных массивов : фиксированный размер и ошибка записи за пределы строки. Для работы с этим классом необходимо подключить заголовочный файл **cstring.h**. В строковом классе реализован большой набор конструкторов для создания строковых объектов, операций и функций для их обработки   
   (**см.** **Help–> Class Library Reference –> C++ Service Classes –> string class**).
7. Применение классов для создания ***защищенных*** массивов позволяет успешно решить эту проблему. Необходимо просто создать класс, в котором выделяется память для данных массива и перегрузить в этом классе оператор индексации []. В функции **operator [] (int)** можно перехватывать индекс, выходящий за рамки диапазона массива и обрабатывать ошибки индексации различными способами.