**Вопросы к экзамену по курсу**

**«Основы программирования на языке C++»**

***Введение в программирование. Переменные и типы данных***

Алгоритм – последовательное описание действий (инструкций, микродействий, команд) созданные для определенного исполнителя, совокупность которых позволяет от исходного состояния (данных) достичь определенного результата.

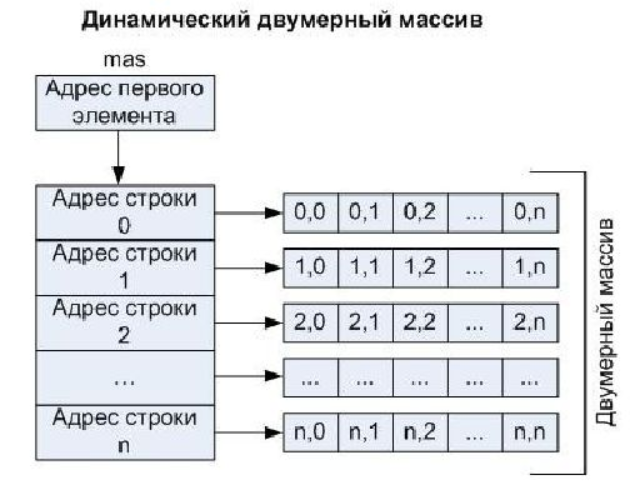
1. Какие существуют виды алгоритмов?  
   - Линейные  
   - Циклические   
   - С ветвлениями (условиями)
2. Опишите 6 фундаментальных свойств алгоритма.  
   - Дискретность (разбиение на шаги)  
   - Детерминированность (выполнение каждого шага одним способом)  
   - Понятность (инструкций для исполнителя)  
   - Массовость (универсальность)  
   - Конечность  
   - Результативность  
   - Формальность
3. Приведите примеры различных типов блок-схем. Для чего они используются?  
   Способы описания алгоритмов:  
   - документная/словесная  
   - псевдокод  
   - графическая  
   - программный код  
   Элементы графических блок-схем:  
   - начало/конец  
   - ввод/вывод данных  
   - действия над данными  
   - условия  
   - цикл
4. Что такое язык программирования?  
   Формальный язык для записи компьютерных программ. Конкретный язык определяет набор синтаксических, лексических и семантических правил.  
   Классификация:  
   - уровень (низкий/высокий)  
   - безопасность (по возможности повредить данные в памяти)  
   - компилируемый/интерпретируемый  
   - системы типов (сильные/слабые)  
   - системы типов (динамические/статические: полиморфные/неполиморфные, явные/неявные)
5. В чём отличие компилируемых языков от интерпретируемых? К какой группе относится C++?  
   Компиляция - исходный код программы преобразуется в целевой (машинный) код специальной программой (компилятором) - в результате получается исполнимый модуль, который уже может быть запущен на исполнение как отдельная программа.   
   Интерпретация же означает, что исходный код выполняется непосредственно, команда за командой, так что программа просто не может быть запущена без наличия интерпретатора.  
   С++ - компилируемый язык
6. Какая типизация данных определена в языке программирования C++ (статическая или динамическая)? Расскажите про особенности различных видов типизации данных.  
   **С++ является статически** типизированным языком программирования. То есть если мы определили для переменной какой-то тип данных, то в последующем мы этот тип изменить не сможем.  
   После первого объявления переменной вы не сможете изменить ее тип позже. Однако можно скопировать значение переменной или возвращаемое значение функции в другую переменную другого типа. Такие операции называются преобразованиями типов, которые иногда необходимы, но также являются потенциальными источниками потери или неправильности данных.  
   Существует автоматическое определение типов.  
   Статические и динамические языки построены на фундаментально разных идеях о корректности программ.  
   **Динами́ческая типиза́ция** — приём, используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов. Примеры языков с динамической типизацией — Smalltalk, Objective-C, Ruby, PHP, Perl, JavaScript, Лисп.
7. Каковы возможные отрицательные последствия подключения пространства имён std?  
   Возможно возникновение конфликта идентификаторов (в том числе в будущем при обновлении std в последующих редакциях языка)
8. Как именно устанавливаются границы диапазонов для разных типов данных?  
   Границы допустимых значений устанавливаются из отведенного кол-ва памяти для каждого типа. Int – 4 байт => 4\*8=32 бит => диапазон +/- 2^32
9. Какие типы данных занимают меньше всего места в оперативной памяти?  
   Те что используют 1 байт – например char
10. Почему наблюдается асимметрия значений границ диапазонов целочисленных типов данных?

Для внутреннего представления знаковых целых чисел характерно определение знака по старшему биту (0 – для положительных, 1 – для отрицательных). Поэтому число 0 во внутреннем представлении относится к положительным значениям. Следовательно, наблюдается асимметрия границ целых промежутков.

1. Чему будет равно значение операции инкремента для максимального числа в целочисленном типе? А каков результат декремента для минимального значения в таком же типе?  
   Целочисленные типы представляют собой кольцо значений, соответственно int\_max++ = int\_min, int\_min-- = int\_max
2. Почему запись целых чисел нельзя начинать с нулей?  
   Воспринимается в С++ (может еще где-то, видимо от реализации стандарта/компилятора) как число с базой 8 (0х – база 16, просто любая цифра кроме 0 – база 10)
3. Опишите правила именования переменных.  
   Можно использовать только латиницу, верх и низ регистра, цифры, \_землю.  
   Начинаться с цифр не может
4. Какими способами можно назвать переменную, состоящую из нескольких слов?  
   Используя\_землю
5. Что будет если к целочисленному типу данных добавить приставку unsigned?  
   Этот префикс при объявлении переменной подразумевает содержание только положительных чисел, при этом расширяя диапазон на (х2-1) элементов
6. Что произойдёт, если значение отрицательной знаковой переменной int передать в переменную беззнакового типа unsigned int?  
   Если в яном виде: С++: ошибка компиляции  
   Если выражением: явление закольцованный диапазона
7. Почему в C++ тип char считается числовым типом данных? Каким диапазоном он обладает?  
   char – базовый тип, 1 байт, диапазон 0-255
8. Можно ли присвоить символьной переменной числовое значение? Ответ аргументируйте.  
   Можно, отображение символов происходит в соответствии с кодом, хранимым в переменной на основании таблицы ASCII
9. Можно ли в символьную переменную char поместить escape последовательность? Ответ обоснуйте.  
   Вроде можно. Char 8..10
10. Допустима ли операция сравнения символов? Если да, то каким образом работают сравнения “больше” и “меньше”?  
    Можно, работает через сравнение кода символа

***Массивы***

1. Почему в программе на языке C++ необходимо, чтобы был известен размер массива?  
   Дабы выделить в памяти (heap/static) необходимое последовательное пространство адресов
2. Каким образам необходимо инициализировать объекты перечисляемого типа (массивы, структуры)?  
   type (int, double, enum) <name> [] {value1, value2, value3};  
   type (int, double, enum) <name> [integer const] {value1, value2, value3};  
   type (int, double, enum) <name> [integer const] = {value1, value2, value3};
3. Можно ли при создании статичного массива передать в качестве длины массива сумму двух констант? Ответ поясните.  
   Можно, это все равно остается константой
4. Даны два массива одинаковой длины A и B. Возможно ли прямое присваивание содержимого массива A в массив B в виде "int B[5] = A"?  
   Вроде нет, только если B[] не был бы массивом указателей, но он int
5. Возможно ли создание безразмерных массивов?  
   Нет, при объявлении в любом случае необходимо указать размер, даже для динамического
6. Может ли значение элемента массива использоваться в качестве индекса другого элемента массива?  
   Мне кажется да, но надо учесть что индекс должен иметь тип приводимый к integer (в умных статьях говорят даже что к типу size\_t)
7. Какие ограничения распространяются на тип массива?  
   Запрещен массив ссылок, функций, const в типе
8. Каким образом можно определить объём памяти, выделяемой под одномерный массив?  
   <кол-во памяти на элемент> \* <кол-во элементов>
9. Каким образом можно определить объём памяти, выделяемый под двумерный массив?  
   Статический: все элементы последовательно в памяти => Объем памяти = <кол-во памяти на элемент> \* <кол-во элементов (в строке)>\*< кол-во строк (массивов, элементов в столбце>  
   ??? как/где хранится указатель на [0][0]?  
   ??? являются ли элементы [0][n] указателями, где хранятся?  
     
   Динамический: <указатель на сам 2-DIM массив: 8 байт > + rows\*<указатель на одномерный массив: 8 байт> + <кол-во памяти на элемент> \* <кол-во элементов (в строке)>\*< кол-во строк (массивов, элементов в столбце>



1. Опишите программу, которая заполняет массив случайными числами из заданного диапазона.  
   srand(time(NULL));  
   any Range\_min, Range\_max;  
   any\* Array = new any[Size];  
   double k\_random = rand()/RAND\_MAX;  
   for (int i=0; i<Size; i++)  
   Array[i]=Range\_min+( Range\_max - Range\_min)\*k\_random;
2. Почему в алгоритме циклического сдвига элементов массива важен порядок смещения элементов?  
   ???
3. Опишите цикл, который перебирает каждый второй элемент массива.  
   for (int i=0 ; i<Size/2 ; i++)  
   arr[2\*i] = <some actions>;  
   //if (Size/2) arr[Size/2+1] = <some actions>;
4. Чем отличаются алгоритмы поиска позиции первого максимального (или минимального) элемента массива от поиска последнего?  
   направлением поиска => ++ либо -- индекса
5. За что отвечает необязательный параметр begin в функциях линейного поиска? В чём разница в инициализации этого параметра для функций, реализующих поиск первого вхождения элемента и последнего?  
   begin – задает начальный элемент поиска (либо от него вперед, либо назад – при поиске последнего)
6. Почему существует большое количество алгоритмов сортировок?  
   Множество алгоритмов сортировки коррелирует с множеством {требований к программе + внешнему виду кода + личных предпочтений }  
   https://imgur.com/gallery/voutF
7. С какой целью используются простые сортировки, если они характеризуются малой эффективностью?   
   Используются на малых объемах данных, для обучающих целей
8. Чем отличаются алгоритмы сортировок по возрастанию от алгоритмов сортировок по убыванию?   
   Алгоритмом действий при наступлении условий сравнения элементов: a[i]>a[i+1] =>
9. В чём отличие алгоритмов сортировок двумерных и одномерных массивов?  
   В задачах сортировок двумерных массивов упорядочиваются по указанному ключу одномерные массивы, из которых построены матрицы.
10. Можно ли создать многомерный массив, не указывая размеры одного из измерений?  
    Можно а[i][]= {};
11. Опишите геометрическую визуализацию трёхмерного массива.  
    куб
12. В чём неудобство использования массивов слишком больших измерений в программах?   
    Массивы слишком больших измерений неудобны в использовании, так как громоздки в объявлении и обращении, их индексация требует дополнительных затрат памяти, что увеличивает время выполнения программы. Задачи перестановок в двумерных массивах – это тип задач, предполагающий обмен значениями элементов массива в зависимости от условия.
13. При каких прикладных задачах используются многомерные массивы?  
    Когда есть несколько множеств с взаимными отношениями - например, транспортная задача: есть много поставщиков и много потребителей, стоимости доставки между разными поставщиками и потребителями разные, надо найти оптимальное распределение - вот вам и двумерный массив стоимостей. Добавьте разные стоимости для разных видов товаров - и массив легким движением руки становится трехмерным.

**Функции**

1. Что такое функция?  
   Функции — это блоки кода, выполняющие определенные операции. Если требуется, функция может определять входные параметры, позволяющие вызывающим объектам передавать ей аргументы. При необходимости функция также может возвращать значение как выходное. Функции полезны для инкапсуляции основных операций в едином блоке, который может многократно использоваться. В идеальном случае имя этого блока должно четко описывать назначение функции.
2. Чем параметры функции отличаются от аргументов функции?  
   Одно и то же - значения, которые функция принимает в круглые скобочки. Значения, с которыми она вызывается/которые ей передаются при вызове. Есть условное разделение, что параметры - это то, что мы прописали в прототипе/объявлении, а аргументы - это то, что функции передаётся фактически в конкретном случае.
3. Можно ли использовать переменную N, которая была создана в main, в теле функции? Почему?  
   Область видимости ограничена пространством функции
4. Можно ли в вызов функции помещать вызов другой функции? Ответ поясните.   
   Можно. Сначала будет вызвана функция внутри вызова, произведет свои действия, потом отработает функция, которая вызывалась изначально
5. Отличается ли механизм вызова встраиваемой функции от обычной? Если да, то как?   
   Да, inline, если позволяет контекст, будет встроена как часть кода прямо в месте вызова, если нет – то как обычная функция это отдельный объект
6. На каком этапе выполнения программы происходит встраивание кода встраиваемой функции?   
   На этапе компиляции
7. Какие могут быть причины ошибки при попытке встроить функцию?  
   Помните о встраиваемых функциях, но современные компиляторы должны встраивать функции за вас по мере необходимости, поэтому нет необходимости использовать ключевое слово inline в этом контексте.
8. Что происходит с функцией, если её не удаётся встроить?   
   Она обрабатывается как обычная функция, без встройки
9. Каким образом компилятор определяет, какую из перегруженных функций необходимо вызвать в программном коде?   
   По лучшему соответствию набору параметров
10. Могут ли перегруженные функции возвращать результат одного типа? Ответ обоснуйте.  
    Могут, перегруженные функции не различаются по типу возвращаемого значения
11. Для чего в шаблонной функции необходимы параметры шаблона?  
    Чтобы компилятор мог отличить для какого типа данных создавать вариант (экземпляра) реализации шаблонной функции
12. На каком этапе выполнения программы создаётся экземпляр функции шаблонной функцией?  
    На этапе компиляции
13. Что происходит в программе, если вызвать шаблонную функцию с одним и тем же набором параметров несколько раз?  
    Ничего особенного, используется несколько раз созданный экземпляр, как в случае обычной функции
14. С какой целью в параметрах функции, которая обладает набором параметров по умолчанию, можно размещать обязательные параметры?  
    Ну, э, видимо часть параметров должна быть задана в явном виде, без возможности задать их по умолчанию
15. Можно ли в списке параметров функции разместить параметры по умолчанию перед обязательными параметрами? Ответ обоснуйте.  
    Низзя, возникает неоднозначная ситуация
16. Сохраняются ли изменения, которые программа выполняет с массивами, переданными в функции? Почему?  
    Сохраняются, потому что передается не сам массив, а указатель на начало и меняются значения в памяти
17. Можно ли в функции с типом void использовать оператор return? Если да, то с какой целью?  
    Завершение работы функции
18. Можно ли создать константный параметр функции со значением по умолчанию?  
    Да, можно
19. Приведите примеры рекурсивных объектов или явлений из реальной жизни. Обоснуйте проявление рекурсии.  
    ???  
    Числа Фибоначчи,
20. Почему при правильной организации рекурсивные вызовы не зацикливаются?   
    Потому что предусмотрено условие выхода
21. Почему при многократном вызове рекурсивной функции одинаковые идентификаторы (названия переменных) не конфликтуют друг с другом?  
    После каждого завершения работы функции перед следующим ее запуском все переменные очищаются (закрывается область видимости)
22. Почему выполнения рекурсивных функций завершается в обратном от вызова этих функций порядке?   
      
    ???
23. Чем ограничено при выполнении программы количество вызовов рекурсивной функции?  
    Ограничено размером стэка
24. Какой из методов в программировании является более эффективным – рекурсивный или итерационный? Ответ обоснуйте.  
    Итерационный, не тратятся ресурсы за вызов функций

**Указатели**

1. Почему указатель не может существовать как самостоятельный тип данных?  
   Указатель указывает на область памяти, а размер области определяется типом данных
2. С какой целью в программе может быть применён указатель типа void?  
   Кажется изящным способом заставить одну функцию обрабатывать несколько типов данных, C++ на самом деле предлагает гораздо лучший способ сделать то же самое (с помощью перегрузки функций), который сохраняет проверку типов, чтобы предотвратить неправильное использование. Многие другие места, где когда-то использовались указатели void для обработки нескольких типов данных, теперь лучше делать с помощью шаблонов, которые также предлагают строгую проверку типов.
3. Как изменится значение указателя при применении к нему операции инкремента или декремента?  
   Значение адреса меняется на <тип данных> байт. При вычислении результата арифметического выражения с указателем компилятор всегда умножает целочисленный операнд на размер объекта, на который указывает указатель. Это называется масштабированием.
4. В чём отличие указателя-константы от указателя на константу?  
   **0) (<data type> / <data type> ptr ):**   
   Обычная ситуация.  
   **Можно все**  
   **1) (const <data type> / <data type> ptr):**const int var = 5;   
   int\* p\_var = &var;   
   **Ошибка  
   1\*) (<data type> / const <data type> ptr):**int var = 5;   
   const int\* p\_var = &var;  
   **Работает.**  
   Нельзя поменять значение \*p\_var  
   Можно перенаправить указатель p\_var = nullptr;  
   Можно изменить var = 10;  
   **2) (const <data type> / const <data type> ptr)**const int var = 5;   
   const int\* p\_var = &var;  
   **Работает.**Нельзя поменять значение \*p\_var  
   Можно перенаправить указатель p\_var = nullptr;  
   Нельзя изменить var = 10;  
   **3) (<data type> / <data type> const ptr)**int var = 5;  
   int const p\_var = &var;  
   **Работает.**Можно поменять значение \*p\_var.  
   Нельзя перенаправить указатель p\_var = nullptr.  
   Можно изменить var = 10;  
   **4) (const <data type> / const <data type> const ptr)**const int var = 5;  
   const int const p\_var = &var;Нельзя поменять значение \*p\_var.  
   Нельзя перенаправить указатель p\_var = nullptr.  
   Нельзя изменить var = 10;
5. Два указателя указывают на одно и то же место в памяти. Изменится ли значение разыменования одного из них, если мы поменяет значение разыменования второго?  
   Да
6. Можно ли применять арифметику указателей к массивам? Ответ обоснуйте.  
   Можно, имя массива – это и есть указатель
7. Можно ли создавать указатели на функции?  
   Можно  
   int (\*fcnPtr)();  
   int (\*const fcnPtr)();
8. Можно ли применять арифметику указателей к указателям на функции?  
   Вроде нет, надо знать кол-во памяти на элемент
9. Что будет если из значения указателя (без разыменования) вычесть значение переменной?  
   Зависит от типа переменной:  
   - целочисленный тип: в соответствии с арифметикой указателей  
   - указатель того же ранга: смещение типа long int  
   - указатель другого ранга: ошибка компиляции  
   - вещественный тип: ошибка компиляции
10. Можно ли вычитать или прибавлять к указателям вещественные значения?  
    Нет, только целые значение
11. Почему в C++ ошибки, связанные с некорректным использованием указателей, являются наиболее трудноустранимыми?  
    Они не видны компилятору  
    Таким образом, после выполнения оператора delete мы получаем недействительный указатель (dangling pointer), указатель, который адресует несуществующий объект. Разыменование такого указателя и последующий доступ к объекту в лучшем случае приведёт к немедленному аварийному завершению работы программы, в худшем — к её нестабильному поведению, которое не повторяется от одного запуска программы к другому. На выявление подобных ошибок в реальных программах зачастую уходит много времени и сил. Поэтому в тех случаях, когда планируется дальнейшая работа с указателем на удаляемый объект, имеет смысл сразу после удаления объекта присвоить указателю на него значение NULL, тем самым явно идентифицировав этот указатель как ни на что не указывающий.
12. Может ли функция возвращать значения типа указатель? Если да, то опишите, как будет выглядеть данная функция.  
    template <typename any> any\* pointer\_return (any arr[])  
    {return arr;}
13. Могут ли параметры функции быть указателями? Если да, то опишите, как происходит передача аргументов в такую функцию. Если нет, то опишите, почему.  
    Могут:   
    int arr[5];  
    pointer\_return(arr);
14. В каких ситуациях могут использоваться указатели на указатели?  
    Многомерные массивы
15. В чём разница между 0, NULL и nullptr?  
    Между 0 и NULL только в этикете, в целом это целочисленный литерал ноль  
    nullptr – особый тип данных nullptr\_t, только для указателей
16. Для чего в программе необходима инициализация указателя перед его использованием?  
    Иначе он никуда не указывает, вызывает неопределенное поведение, ошибка на этапе выполнения
17. Опишите последствия использования в программах неинициализированных указателей.  
    Неопределённое поведение. Попытка получения доступа по мусорному значению в памяти  
    ??? Почему вообще при создании указателя он не nullptr, зачем там по умолчанию мусор?
18. Что будет если разыменовать нейтральный указатель?  
    Неопределённое поведение, отказ в доступе к нулевым адресам, ошибка на этапе выполнения
19. В чём отличие обращения к элементам массива с помощью индексированного имени и посредством арифметики указателей?  
    Разница только в синтаксисе, в результате разницы нет  
    arr[5] = \*(arr +5), &arr[5] = arr+5
20. Эквиваленты ли для массива arr следующие записи: arr и &arr[0]?  
    Очень похоже что да
21. Эквивалентны ли для двумерного массива mx следующие обращения и почему: mx и &mx[0][0]?  
    Да, указывает на первый элемент двумерного массива, он же – первый элемент первого вложенного массива
22. Возможно ли в двумерном массиве mx обращение к элементу &mx[0]?  
    Да – это адрес указателя на первый вложенный массив
23. Опишите обращение к элементу arr[i][j] с помощью арифметики указателей.  
    arr[i][j]   
    \*(arr[i] + j)  
    \*(\*(arr+i) + j)
24. Опишите обращение к элементу arr[i][j][k] с помощью арифметики указателей.  
    \*(arr[i][j] + k)  
    \*(\*(arr[i] + j) + k)  
    \*(\*(\*(arr + i) + j) + k)
25. В чём отличие ссылки от указателя?  
    указатель – переменная, хранящая адрес  
    ссылка – просто адрес, как псевдоним переменой  
    Нельзя объявить массив ссылок.

У ссылки нет адреса.

Существует арифметика указателей, но нет арифметики ссылок.

1. Какой размер ссылки в байтах?  
   0
2. Как компилятор реализовывает работу ссылок?  
   Зависит от конкретной реализации компилятора:  
   - можно через непосредственное обращение к переменной  
   - можно через указатель  
   На мой взгляд очень похоже на работу константного указателя  
   Предлагается считать абстрактно что это создание псевдонима (второго имени) для переменной
3. Опишите области применения ссылок.  
   Везде где можно советуют применять ссылки, это проще синтаксически и оставляет меньше возможностей для ошибок.  
   Где нельзя применить ссылки вместо указателей:   
   - создание массивов из указателей только может быть
4. В чём отличие передачи данных в функцию по значению, адресу и ссылке?  
   - по значению: создается копия для работы внутри функции с копией  
   - по адресу: через передачу указателя в функцию передается адрес аргумента, далее можно работать с значением через обращение к адресу (\*)  
   - по ссылке: почти то же что через указатель, но синтаксис проще и вероятность ошибок меньше

**Динамические массивы**

1. Для чего используется динамическая память в программировании?  
   Для динамического выделения, т.е. использование неизвестного в момент компиляции кол-ва памяти: в первую очередь динамические массивы, управление временем жизни объектов, размещение больших объектов (не влазящих в стэк)
2. Какая область выделяется под размещение динамических данных?  
   Динамическая (в отличии от статической и автоматической (стека))  
   **Статическая память** — это область памяти, *выделяемая при запуске программы* до вызова функции *main* из свободной оперативной памяти для размещения глобальных и статических объектов, а также объектов, определённых в пространствах имён.  
   **Автоматическая память** — это специальный регион памяти, *резервируемый при запуске программы* до вызова функции *main* из свободной оперативной памяти и *используемый в дальнейшем* для размещения локальных объектов: объектов, определяемых в теле функций и получаемых функциями через параметры в момент вызова. Автоматическую память часто называют **стеком**.  
   Размещение объектов в автоматической памяти и их удаление из неё происходит по принципу **LIFO –** отсюда название **стэк**

**Динамическая память** — это совокупность блоков памяти, выделяемых из доступной свободной оперативной памяти непосредственно *во время выполнения программы* под размещение конкретных объектов.

1. Как долго данные хранятся в динамической памяти?  
   До завершения работы программы, либо до применения delete
2. Какие возможны варианты доступа к динамической памяти?  
   **С объектом в куче мы можем работать (получать или изменять его состояние) только посредством указателей.**  
   Возможны три варианта работы с динамической памятью: указатель определяется как локальный объект автоматической или статической памяти, указатель является глобальным объектом для блока памяти.
3. Что возвращает операция выделения динамической памяти в случае успешного выполнения?  
   Начальный адрес элемента, под который была успешно выделена динамическая память
4. Что возвращает операция выделения динамической памяти, если участок требуемого размера не может быть выделен?  
   Ранее, до появления в стандарте C++ исключений (exceptions), в таких случаях оператор new возвращал значение NULL. С появлением исключений поведение оператора new было изменено — теперь при нехватке памяти он генерирует исключение std::bad\_alloc.
5. Почему при завершении работы с динамической памятью её необходимо освободить? Какие могут быть последствия для работы программы, если не освобождать динамическую память?  
   1) Просто освобождение не используемой памяти  
   2) Избежание возможных утечек памяти при перенаправлении указателя  
    3) Избегание возникновения недействительных указателей (dangling pointer)
6. Существуют ли ограничения на данные при применении к ним операции или функции освобождения динамической памяти?   
   Операцию delete следует использовать только для указателей на *память*, выделенную с помощью *операции* new. Использование delete с другими типами адресов может породить серьезные проблемы.
7. В каких ситуациях в программировании целесообразно использовать динамические массивы?  
   - неизвестное кол-во элементов  
   - большое кол-во элементов
8. Что будет возвращено при попытке объявить динамический массив недопустимо большого размера?  
   - Если инициализировать литералом:  
   double\* pointer2 = new double[10e8];

то:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Severity | Code | Description | Project | File | Line | Suppression State |
| Error (active) | E0095 | array is too large | Testing | C:\Users\Papa\source\repos\Testing\Testing\Source.cpp | 7 |  |

- Если инициализировать выражением:  
long long index = 10e9;

double\* pointer = new double[index];

то:  
Unhandled exception at 0x00007FF97AFC051C in Testing.exe: Microsoft C++ exception: std::bad\_alloc at memory location

int\* a1 = new int[SIZE]; // SIZE limited only by OS/Hardware

кроме того, если массив выделяется на стеке, то вы ограничены размером кадра стека. **Б. Н.** векторы и другие контейнеры имеют небольшое присутствие в стеке, но обычно основная часть данных будет находиться в куче.

int a2[SIZE]; // SIZE limited by COMPILER to the size of the stack frame

1. Как размещаются в памяти элементы одномерного динамического массива?  
   Переменная указатель (имя массива) хранится где-то в стеке и содержит адрес первого элемента массива  
   Элементы – последовательно.
2. Какими способами можно обратиться к элементам одномерного динамического массива?  
   - через синтаксис массивов arr[i];  
   - через арифметику указателей и обращение по адресу \*( arr+i);
3. С какой целью используются квадратные скобки в операции освобождения динамической памяти, выделенной под массив: delete [] mass?  
   Чтобы вызвать именно функцию удаления динамического массива, а не переменной
4. В чём сходство и отличие одномерных и двумерных динамических массивов?  
   сходство: элементы располагаются последовательно  
   различия: каждая новая размерность порождает новый указатель N-порядка на массив указателей (N-1)-порядка
5. Как размещаются в памяти элементы двумерного динамического массива?  
   - Указатель на динамический массив указателей (одномерный). Вероятно размещается на стеке. Является именем массива  
   - Динамический массив указателей на одномерные динамические массивы элементов (первых элементов). Этот массив находится в динамической области, располагается последовательно из 8 байтовых (х64) целочисленных элементов  
   - Элементы двумерного массива располагаются последовательно в динамической памяти
6. Что является значением двойного указателя при объявлении двумерных динамических массивов?  
   Адрес первого элемента одномерного массива указателей на одномерные массивы элементов
7. Как выделяется память для двумерных динамических массивов?  
   Через цикл  
   double\*\* arr = new double[rows];  
   for (int i=0; i<rows;i++)  
   arr[i]=new double[columns]; //как получается что элементы хранятся последовательно, если можно объявлять вложенные массивы не сразу  
   {…}  
   for (int i=0; i<rows;i++)  
   delete [] arr[i];  
   delete[] arr;
8. Какими способами можно обратиться к элементам двумерного динамического массива?  
   - синтаксис массивов: arr[i][j]  
   - синтаксис арифметики указателей: \*(\*(arr+i)+j)  
   - синтаксис массивов + синтаксис арифметики указателей: \*(arr[i]+j)
9. Почему ошибки при работе с динамической памятью относят к опасным?

Таким образом, после выполнения оператора *delete* мы получаем *недействительный* указатель (dangling pointer), указатель, который адресует несуществующий объект. Разыменование такого указателя и последующий доступ к объекту в лучшем случае приведёт к немедленному аварийному завершению работы программы, в худшем — к её нестабильному поведению, которое не повторяется от одного запуска программы к другому. На выявление подобных ошибок в реальных программах зачастую уходит много времени и сил. Поэтому в тех случаях, когда планируется дальнейшая работа с указателем на удаляемый объект, имеет смысл сразу после удаления объекта присвоить указателю на него значение NULL, тем самым *явно* идентифицировав этот указатель как ни на что не указывающий.

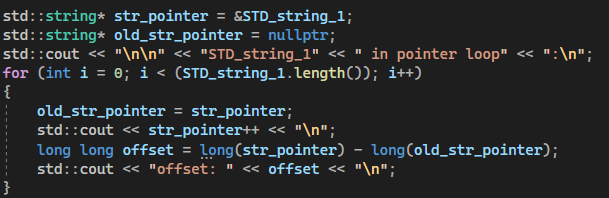
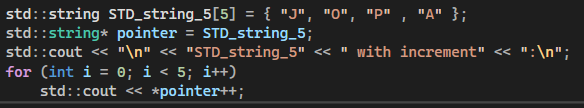
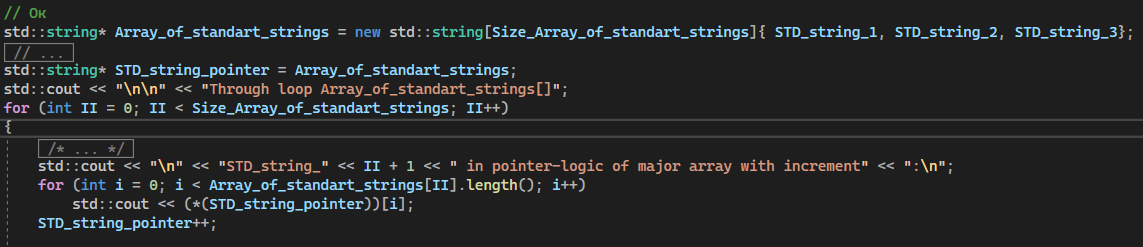
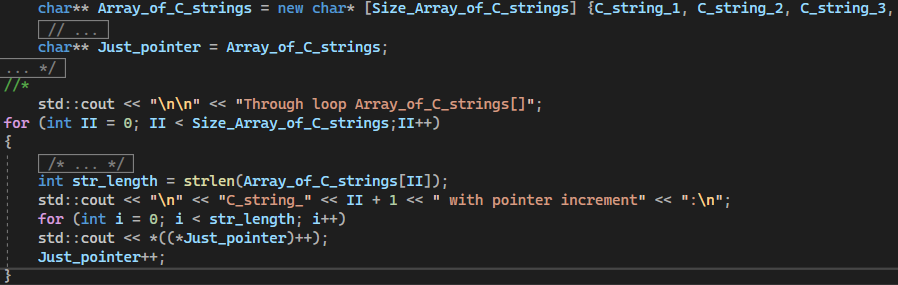
**Строки**

1. Что необходимо подключить в программу для работы со строковым типом данных string?  
   заголовочный файл <string>
2. Почему символ и строка, состоящая из одного символа, занимают разный объём памяти?  
   Объект типа std::string хранит символ окончания строки
3. Почему при сравнении строк важен регистр символов?  
   Коды символов заглавных букв больше прописных
4. Как сравниваются строки разной длины?  
   операнды сравниваются посимвольно до тех пор, пока не будет найдена пара соответствующих символов, которые содержат разные символы, или пока не будет достигнут конец одного или обоих операндов
5. Как сравнить длины двух строк?  
   .length() .size()
6. Почему введённая на кириллице через консоль строка может некорректно отображаться при попытке вывести её?  
   1. Windows извращенно-изощренная сущность  
   2. Для всех языков использующих не латиницу существуют различные кодовые таблицы (потому что памяти пожалели и выделили 1 байт всего на 255 символов)  
   3. Консоль использует разные кодовые страницы в буферах ввода и вывода  
   4. Кроме того консоль реагирует на переключение «локали» - глобальные языковые настройки системы
7. Что происходит при обращении к неинициализированной строке?  
   Ничего особенного, это просто строка пустая строка “”, но некоторые функции, например stod/stoi выбрасывают исклечения
8. Можно ли применить функции изменения регистра к строке целиком?  
   - как будто без дополнительных подключений нет – только посимвольно:  
   string[i]=tolower(string[i]);  
   - Boost  
   #include <boost/algorithm/string.hpp>

std::string str = "HELLO, WORLD!";

boost::algorithm::to\_lower(str); // modifies str

1. Почему при передаче в функцию строки, которая является символьным массивом, передаётся адрес первого символа строки?   
   Стандартная работа с указателем
2. Возможно ли применение операций инкремента и декремента к указателю на строковый массив типа char? А на массив типа string?  
   Указатель на строковый массив типа char:  
   - char\* C\_string {“TEXT”}  
   - char\* C\_string\_array[] {“TEXT1”, “TEXT2”}  
   обращение через указатель возможны, в том числе с декрементом  
     
   Указатель на строковый массив типа string:  
   - std::string\* Array\_of\_standart\_strings = new std::string[3]{ STD\_string\_1, STD\_string\_2, STD\_string\_3}; вот такой указатель можно перебирать декрементом  
     
     
   - std::string\* str\_pointer = &STD\_string\_1; вот такой тоже можно декрементировать, но прибавляет по 40 байт сразу, и выходит за границу разрешенного доступа, так что посмотреть нельзя



1. За что отвечает escape-последовательность '\0'? Почему её не обязательно указывать при создании символьного массива?  
   Строка в стиле С это массив символов с завершающим нулем:  
   char C\_string[] = “text\_value”

char\* C\_string = new char[Size+1]

const char C\_string[] = “text\_value” – здесь будет в конце \0

char C\_string[] = {‘t’, ‘e’, ‘x’, ‘t’} – а здесь не будет в конце \0

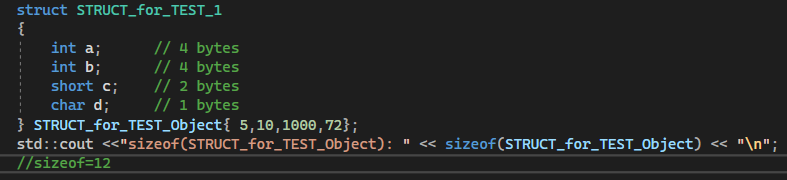
1. Почему cout при обращении к строковому массиву выводит всю строку целиком, а при обращении к массивам других типов выводит какое-то шестнадцатеричное число?  
   Видимо есть перегрузка << для работы с массивами char в cout – воспринимать их как строки, а значения – как коды ASCII, длина до ‘\0’

**Структуры**

1. В чём отличия массива от структуры? Перечислите минимум 4 отличия.

| **МАССИВ** | **СТРУКТУРА** |
| --- | --- |
| Массив относится к коллекции, состоящей из элементов однородного типа данных. | Структура относится к коллекции, состоящей из элементов разнородного типа данных. |
| Массив использует индексы или “[ ]” (квадратная скобка) для доступа к элементу | Структура использует “.” (оператор точки) для доступа к элементу |
| Массив является указателем, поскольку он указывает на первый элемент коллекции. | Структура не является указателем |
| Создание экземпляров объектов массива невозможно. | Возможно создание экземпляра объектов Structure. |
| Размер массива фиксирован и в основном представляет собой количество элементов, умноженное на размер элемента. | Размер структуры не фиксирован, поскольку каждый элемент структуры может быть разного типа и размера. |
| Битовое поле невозможно в массиве. | В структуре возможно битовое поле. |
| Объявление массива выполняется просто с помощью [], а не какого-либо ключевого слова. | Объявление структуры выполняется с помощью ключевого слова “struct”. |
| Массивы - это непримитивный тип данных | Структура - это определяемый пользователем тип данных. |
| Обход массива и поиск выполняются легко и быстро. | Обход структуры и поиск являются сложными и медленными. |
| data\_type имя\_данных[размер]; | имя\_структуры{ data\_type1 ele1; data\_type2 ele2; }; |
| Элементы массива хранятся в смежных ячейках памяти. | Элементы структуры могут храниться, а могут и не храниться в непрерывной области памяти. |
| Доступ к элементам массива осуществляется по их индексному номеру с использованием индексов. | Доступ к элементам структуры осуществляется по их именам с помощью оператора dot. |

1. Как происходит мгновенная инициализация объекта структуры, в котором одно из полей является объектом другой структуры?  
   Для вложенных структур вы можете использовать вложенные списки инициализаторов:  
   Company myCompany{{ 1, 42, 60000.0 }, 5 };
2. Как располагаются в памяти элементы структуры?  
   Элементы структуры могут храниться, а могут и не храниться в непрерывной области памяти.
3. Почему размер структуры не всегда совпадает с суммарным размером её полей?  
   Оказывается, мы можем сказать только то, что размер структуры будет не меньше размера всех содержащихся в ней переменных. Но может быть и больше!



1. Для чего создано выравнивание структур?  
   По соображениям производительности компилятор иногда добавляет разрывы в структуры (это называется заполнением).  
   Одной из низкоуровневых особенностей C++ является возможность указать точное выравнивание объектов в памяти, чтобы максимально использовать конкретную аппаратную архитектуру. По умолчанию компилятор выравнивает члены класса и структуры по их значению размера: bool и char по 1-байтовой границе, short по 2-байтовой границе, int, longи float по 4-байтовой границе и long long, doubleи long double по 8-байтовой границе.
2. Для моделирования каких данных целесообразно использовать структуры?  
   Для большого кол-ва данных с одинаковыми наборами свойств(полей).  
   Например: товары на складе = struct Goods {sting name; int quantity; char group;} car1{“BMW”, 50, 44};
3. Какими способами можно обратиться к полям объектов структуры?  
   1. Через метод «.»:  
   имя\_объекта\_структуры.имя\_элемента  
   2. Через указатель на структуру:   
   2.1 (\*указатель\_на\_структуру).имя\_элемента  
   2.2 Второй способ предполагает использование операции -> (операция стрелка):

указатель\_на\_структуру->имя\_элемента

1. В чём отличие прямого и косвенного доступа к полям объектов структуры?  
   Синтаксис
2. Даны два объекта одной структуры A и B. Возможно ли прямое присвоение содержимого объекта A в объект B в виде "person B = A;"? Если да, то при каких обстоятельствах это может привести к ошибке?   
   Можно  
   Проблемы могут возникнуть при использовании динамических массивов, указателей
3. Для моделирования каких данных целесообразно использовать массив структур?

БД

1. Какие данные о структуре содержит указатель на объект этой структуры?  
   Адрес начального элемента структуры
2. Какие ограничения накладываются на тип элемента структуры?  
   Массив ссылок  
   Массив функций
3. Возможно ли использование указателя на объект структуры в качестве поля другой структуры?
4. Как выполняется инициализация указателя на объект структуры?  
   sctruct struct\_1 {} struct\_1\_object;  
   struct\_1\* struct\_1\_object\_pointer = &struct\_1\_object;
5. Как выполняется операция инкремента и декремента над указателями на объекты структуры?
6. Каким образом необходимо передать объект структуры в качестве параметра функции, чтобы сохранить изменения, совершаемые функцией над данным объектом?

по ссылке void funс(user\_struct& user\_data){}

1. Что возвращается в качестве результата работы функции, тип которой объявлен как структура?   
   Объект структуры
2. Что возвращается в качестве результата работы функции, тип которой объявлен как указатель на структуру?  
   Указатель на структуру?

**Работа с файлами**

1. Каковы могут быть причины ошибок при открытии файлов?
2. Какие значения возвращает функция is\_open()?
3. С какой целью предусмотрены режимы открытия файлов и почему их такое многообразие?
4. Каким образом в программе происходит проверка достижения конца файла?
5. Может ли один и тот же файл быть одновременно открыт для чтения и для записи?
6. Можно ли один и тот же файл открыть несколько раз, не закрывая его?
7. Сохранится ли информация в файле, если его не закрыть в программе с помощью функции?
8. Возможно ли расположить курсор в файле в начале файла? Если да, то как? Возможно ли расположить курсор в файле в середине файла? Если да, то как?
9. С какой целью выделены отдельные потоки ofstream, ifstream, fstream?
10. В чём отличие выполнения метода open() с одним параметром и с двумя параметрами?
11. Опишите алгоритм посимвольного считывания данных из файла.
12. Опишите алгоритм пословного считывания данных из файла.
13. Опишите алгоритм построчного считывания данных из файла.
14. Как считать данные с определённой позиции файла?
15. В чём отличие метода seekg() от метода seekp()?
16. Опишите алгоритм добавления записи в середину файла.

**Общие вопросы**

1. Сколько раз проверится условие в цикле while по сравнению с циклом do…while с тем же телом цикла?
2. Можно ли оставить поля цикла for пустыми? Что в таком случае будет происходить с циклом? Опишите каждое поле цикла for.
3. Может ли конструкция множественного выбора switch принимать на вход строковые данные?
4. В чём разница между операторами break и continue? Где они применяются?
5. Опишите отличие операции инкремента A++ от прединкремента ++A.
6. В чём разница между строковым массивом и строкой string?
7. Каким образом компилятор реализует механизм работы ссылок?
8. Что будет, если в рамках одного case в структуре switch не указать оператор break?
9. Опишите общий принцип работы директив препроцессора define и include.
10. Что такое defguard и какую проблему он решает?
11. Почему поток ввода-вывода не зависит от конкретного устройства?
12. Для чего необходима буферизация при потоковом и форматированном вводе выводе?