*1.История развития персональных ЭВМ(PC- Personal Computer)*  
1972г.-появл. 8-бит микропроц. Intel 8008. 1974г.-Intel 8880. 1979г.-Intel 8086 с 4,77; 8; 10 МГц. 1981г.- IBM MSDOS 1.0. 1981г. принцип открытой архитектуры. 1982г.-Intel 80286 (16МГц, 16МБ ОЗУ, 1Гб) 1985г.-Inyel 80386(40МГц) 1989г.-Intel 80486(66МГц) 1991г.-MSDOS 5.0. 1992г.-Intel 80486X4 (100МГц) 1992г.-Windows 3.1. 1995г.-Windows 95

*2.Теория алгоритмов*  
Решение люб. задачи на люб. ЯП основывается на том или ином алгоритме. Начальным этапом разработки люб. пр-ммы явл. сост. алгоритма

*3.Понятие алгоритма и его свойства*  
Алгоритм – последовательность команд, предназначенная исполнителю, в результате выполнения которой он должен решить поставленную задачу. Исполнитель – компьютер, автоматизированное устройство, человек, умеющие механически выполнять команды, составляющие алгоритм. Свойства алгоритма: 1) дискретность(состоит из отдельных шагов); 2) понятность(понятные указания для исполнителя); 3)определенность «детерминированность»(1 вариант толкования действий); 4)результативность(решение задачи за конечное число шагов); 5) массивность(решение множества однотипных задач); 6)конечность(завершение алгоритма за конечное число шагов)

*4.Основные этапы алгоритмизации*  
Алгоритмизация – процесс создания алгоритма, состоящего из следующих этапов: 1) разработка(осущ.в соот-ии с прининципами структ.проект.делающего представл.и пониман.алгоритма простым); 2) обоснование(док-во недопущения ошибки и точности рез-та за кон. кол-во шагов); 3) представление(запись алгоритма в виде схемы); 4) анализ и 5) тестирование(док-во правильности алгоритма и его тестирование на различных наборах данных. Ошибки: синтаксические и логические. Тестирование – процесс вып. алгоритма с целью найти ошибки путем реш. тест. задач., рез-т. кот. известен заранее.)

*5.Правила оформления схемы алгоритма*  
ГОСТ 19 701-90. Алг.сост.из:1) Символов 2) Пояснит. текста 3) Соедин. линий. Бывают основные(неизв. тип или вид процесса) и специфические(известен точн. вид или тип процесса). Схема – граф. решение задачи, в кот. исп. фигуры для отобр. операций, данных, действий.

*6.Язык программирования Си. История создания*  
Си (англ. C) — компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах сотрудником Bell Labs Деннисом Ритчи. Связан с операционной системой Unix, но был перенесен и на множество других платформ. Изначально создавался для решения системных задач. Популярность язык завоевал благодаря разделению свойств на машинно-зависимые и независимые. Можно было разработать программу на любой ПК, независимо от процессора и памяти. При этом аппаратно зависимую часть можно было прописать отдельно. 1989г. – разработка спецификации языка С (стандарт ANSI). 1999г. – стандарт С-99 введен орб. Обработки исключений. Язык программирования Си оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования, как C++, C#, Java и Objective-C.

*7.Достоинства и особенности языка Си*  
1) Современный язык, т.к. используется нисходящее программирование, структурное программирование и пошаговая разработка модулей. Результат – надежная и легко читаемая программа;2) Эффективный язык. Программы на Си компактные и быстрые;3) Переносимый и мобильный язык;4) Мощный и гибкий;5) Удобный язык (Маленькое количество ограничений)

*8.Алфавит языка Си*  
Алфавит языка состоит из 4х групп:1) Все буквы латинского алфавита и знак подчеркивания(Строчные используются для написания ключевых слов языка. Одинаковые строчные и прописные буквы имеют разный код и при записи индиф Си различаются. Буквы руского алфавита используются для ввода текста и комент.); 2) Цифры от 0 до 9; 3) Специальные символы(.,;:?!`|/\~+-\*()[]{}<>=&#%””); 4) Управление последовательностью при вводе/выводе инф.

*9.Идентификаторы*  
Это имена, которые обозначают некоторые объекты в программе. Данные в оперативной памяти размещаются по адресам, заранее известных программе. Для того чтобы иметь возможность обращаться и обрабатывать данные в программе, программист дает данным условные имена, которые компилятор заменяет на адрес в памяти. Для записи инф используются только буквы латинского алфавита, цифры и знак подчеркивания. Компилятор различает идентификатор по первым 32 символам. Идентификаторы – составная часть языка Си. Они имеют фиксированный написанный и определенный смысл, не используется для других целей. При выборе идент. учитывают:1) Не должен совпадать с ключевыми словами языка и именами функций; 2) Не рекомендуют начинать с знака «\_» (компилятор автоматически добавляет «\_»); 3) Правильно выбранный идент. облегчает чтение и понимание программы.

*10.Ключевые (зарезервированные) слова*  
Это имена, используемы в Си с заранее определенным смыслом. Они сообщают компилятору информацию о типе данных, способе обработки и прядке выполнения. Ключевые слова: 1) для указания операции получения размера объекта: sizeof; 2) для описания прототипа объекта: typedef; 3) для объявления переменных(для обозначения класса хранения переменных: auto, registerдля обозначения класса хранения переменных: auto, register|для обозначения того, что объект описывается в другом месте extern;| для обозначения того, что объект статический static;); 4) для обозначения типа переменных char, short, int, long, signed, unsigned, float, double, void (для указания на произвольный родовой тип); 5) для обозначения специальных типов данных(struct (структура), enum (перечисление), union); 6) для обозначения операторов и их элементов(для обозначения операторов цикла: do, for, while; условного оператора: if, else; оператора выбора: switch, case, default;); 7) для обозначения операторов перехода(операторы прерывания исполнения кода: break, continue; для обозначения оператора безусловного перехода: goto); 8) для обозначения оператора возврата из функции: return. 9) для обозначения того, что функция является встраиваемой: inline; 10) для обозначения типа переменной: \_Bool, \_Complex, \_Imaginary; 11) для обозначения того, что объявляемый указатель указывает на блок памяти, на который не указывает никакой другой указатель: restrict.

*11.Типы данных языка Си*  
Существуют типы данных и модификации типов. Базовые типы данных(char – символьный; int – целый; float – вещественный; double - вещественный с двойной точностью; void - пустой тип) К модификациям относятся(signed – знаковый; unsigned – беззнаковый; short – короткий; long - длинный) Тип и мод-р определяют( 1) формат хранения данных в ОЗУ; 2) диапозон зн., в пределах кот. может измен. Переменная; 3) операции кот. могут вып. над данными соотв. типа) Типы: 1) Скалярные (символы,арифметические(цел, вещ.), указатели, перечисления) 2) Составные(массив, поля битов, стр-ра, объединения).

*12.Константы и переменные*  
Переменные - это эл-ты данных, зн-я кот. во время вып. пр-ммы можно изменять. Константы – не изменяемые данные. В программе используемые переменные должны быть определены и объявлены. В описании указывается тип данных и через ‘,’ перечисляются имена переменных, имеющих данный тип. В модуле, где записывается определение переменной, для каждой переменной в соответствие с типом данных выделяется необходимое количество памяти. Спецификатор типа – одно или несколько слов, определ тип переменной. Ключевые слова определяют класс памяти (каким образом для переменной распределяется память и где необходимо искать значение переменной)

*13.Данные целого типа*  
Тип int – цело число, обычно соответсвующее естественному размеру целых чисел. Квалификаторы short и long указывают на различные размеры и определяют размеры и объем памяти, выделенный под них. chart(1байт, 0..255) int(2байта, 0..65536) long(4байта,0..4,294,967,295)

*14.Вещественные типы данных*  
Переменные с плав. точкой Float(7 знаков,4байта,1,17E-38..3.37E+38), double(15знаков,8байт,2.23E-308..1.67E+308), long double(19 знаков,10байт,3.37E-4932..1.2E+4932). Сост. из мантиссы m и порядка p. С=m\*2p.

*15.Символьный тип данных*  
Символьная переменная занимает в памяти один байт(от 0…255). Закрепление конкретных символов за кодами производится кодовыми таблицами (ASCII). Данные типа char рассматриваются компилятором как целые и могут быть знаковыми и без знаковыми. Код символа является порядковым номером в таблице. В символах и строках могут использоваться управляющие знаки

*16.«Пустой» тип void*  
Отсутствие описания типа. Описание указателя общего типа, когда заранее извест. тип объекта на кот. он будет ссылаться.

*17.Совместимость типов*  
Типы char и int могут свободно смешиваться в арифм. выр-ях: char преобраз. в int. Преобразования: 1) Если +,\* то низший тип преобр. в высший (сhar→int; unsigned long→float). 2) Если 1 из операндов unsigned, то др. операнд и рез-т также преобраз. в этот тип. 3) При сужающихся преобраз. в целых типах лишние биты более старш. порядков отбрасыв. а дробные типы округл.

*18.Структура простой программы на языке Си*  
Исходники быв. 2 типов: 1) Заголовочный – h-файлы 2) файлы реализации с, сpp, cc, c++. Заголовчные ф-лы сод-т только описание(прототипы ф-ий, число и типы аргументов ф-ий, константы, стр-ры, служат для передачи инф-ии между модулями).Ф-лы реализ. сод-т тексты ф-ий и определения глоб. переменных, и предст. соб. отдельные модули. Вып. люб. пр-ммы нач с ф-ии main. Исходная пр-мма сост. из: директив, указание компилятору, объявлений и опредлений. Препроцессор – пр-мма предварит. обраб. текста перед трансляцией, используется для подключения заголовочных файлов. Директивы – команды препроцессора, сод-т символ # в начале строки. Указание компилятору – команды, вып. комплилятором во время процесса компиляции. Объявления – задают имена и атрибуты переменных ф-й и типов, исп. в пр-мме. Определения – объявления определяющие переменные(в дополн. к имени мож. задавать нач. значение) и ф-ии.

*19.Директивы процессора Си*  
Предст. соб. инструкции в виде текст. пр-ммы, позвол. измен. текст пр-ммы, вставить текст из др. файла и т.д

*20.Директива #include*  
Включает в текст программы указанные файлы и имеет две формы < > и “ ”. Имена файлов должны составлять ОС и могут содержать предметный маршрут. Если используются угловые скобки, то поиск файла будет осуществляться в стандартных территориях среды разработки.

*21.Директива #define*  
Исп. для замены часто использующихся констант, операторов, ключ. слов или выр. некот. идентификаторами. Ид-ры заменяющие текст или числ. конст. наз-ся именовынными константами. Ид-ры замен. фрагмент пр-ммы наз. макроопределения. Данный процесс называется макроподстановка. Текст может представлять собой фрагмент программы на си, а может и отсутствовать.

*22.Директива #undef*  
Исп. для отмены действия #define. #undef ид-р; #undef width.

*23.Комментарии*  
Текст. части для аннотирования пр-ммы. Способы задания: /\*..\*/ -коменты допуск. везде, где разрешенны пробелы. // -в 1-й строке. Компилятор игнорирует текст комментария.

*24.Стандартная математическая библиотека функций*  
<math.h> преобладает тип double, т.е. корректно раб. с типом float, longdouble-приводит к ошибкам. Примеры: abs(intx), fabs(double x), log(double x), log10(double x), exp(double x), sin(double x), floor(double x), hypot(double x, double y), pow(double x, double y).

*25.Операции в языке Си*  
Это спец. комбинации символов выполняющие действия по преобраз. различных типов величин. Компилятор интерпретирует кажд. из этих комбин. как самост. ед-цу наз. лексемой. Все операции делятся на унарные(действие над 1 операндом) и бинарные(над 2-мя).

*26.Арифметические операции*  
+, -, \*, /, %(остаток от деления).  
*27.Операции поразрядной арифметики*  
&(и), |(или), ^(исключающее или), ~(инверсия), <<(сдвиг влево), >>(сдвиг вправо).  
*28.Логические операции*  
&&(и) ||(или) !(отри-цание).  
*29.Операции отношения*  
==(сравнение на равенство), >, <, >=, <=, !=.  
*30.Операции присваивания*  
=, ++(унарный инкримент), --(унарный декремент).

*31.Условная (тернарная) операция*  
&(адресация) \*(разадресация), «,»(послед. вычисление), ? (условное выражение, усл. тернарн. операция).

*32.Операция sizeof*  
Определение размера в байтах. a>sizeoff(int) - определяет размер памяти кот-му соотв. ид-р в памяти или тип a присваивает размер типа int

*33.Операции «увеличить на», «домножить на» и т.п.*  
+= (x+y→x+=y), - = (x=x-y→x-=y), \*=, /=, %=, &&=, ||=.

*34.Операция привидения типа*  
type cast – исп. когда значение 1 типа преобр. к другому в случае, если сущ. разумный способ такого преобразования. Пример 1. int→float: double x; int n; x=(double)n; Пример 2. float→int: double x,y; int n,k; x=3,7; y=(-1,5); n=(int)x; //n=3 k=(int)y //k=-1.

*35.Операции «&» и «\*»*  
& - операция взятия адреса явл. унарной, т.е. переменной y присв. адрес переменной x. y=&x; \* - воспринимает операнд как адрес нек. объекта и исп. его для выборки содержимого. z=\*y; int \*L, float \*p – L,p – указатели.

*36.Выражения. Приоритет операций*  
Операнд-константа либо переменная. ()[] . \* - > (выражение, л→п); - ~ ' \* & ++ -- sizeoff (унарный, п→л); \* / % (мультипликативный, л→п); + - (аддитивный, л→п); << >>(сдвиг, л→п); < > >= <= (л→п); == , ! = (отношение, л→п); & ^ (л→п); ^( л→п); | (побит. вкл. или, л→п); && || (л→п); ?(условная, п→л); = \*= /= %=(прост. и сост., п→л); += -= <<= >>= &= != ^= (присваивание) , (послед. преобраз, л→п).

*37.Функции ввода-вывода в языке Си*  
<stdio.h> printf(“упр. строка”,аргументы1,аргументы2); Упр. строка может содерж. как обычн. символы так и спецификаторы преобразования, управляющие символьные константы. Между спец-ром и символом встречаются: «-» указывает что преобразованный параметр должен быть выравнен влево в своем поле; «.»-отделяет размер поля от след. строки; строка цифр - задает мах размер поля; символ длины цифр-указывает что соотв. аргумент имеет тип long. Символы: d-десят. число, o-восьмеричное, х-шестнадцатеричное, с-символ, s-строка, е-вещ.десят.число, f-вещ.дес.число с плав точкой, g-искл. вывод незначащих нулей, p-знач. аргумента явл-ся указателем, u-беззнаковое число(unsigned).

*38.Функция форматного вывода printf()*  
printf исп. упр. строку чтобы определить ск-ко всего аргументов и каковы их типы. \n-переход на новую строку, \t-горизонтальная табуляция.

*39.Функция puts()*  
Записывает симв. строку в стандартный поток данных(выводит на экран). int puts (const chat \*string). После вып. этой ф-ии курсор переход. в новую строку и ф-ия возвр. код символа

*40. Функция putchar()*  
int putchar (int ch); записывает символ в стандартный поток данных, возвр. выведенный на экран символ

*41.Функция форматного ввода scanf()*  
scanf(“упр.строка”,&aрг1,арг2,...); Аргументы–указатели на соотв. значения. Символы: d-десят. число, o-восьмеричное, х-шестнадцатеричное, с-одиночный символ, s-строка, е-вещ.десят.число, f-вещ.дес.число, p-указатель в виде 16-го числа.

*42.Функция gets()*  
char \*gets(char \*buffer); Счит. симв. строку и помещ. ее по адресу заданному указателем в буфер. Прием строки заканч. на сим

*43.Функция getchar()*  
int getchar(void); Считывает символ из стандартного входного потока и возвр. его.

*44.Операторы языка Си*  
Условные(if-разветвления switch-выбора), организации циклов(while, do..while, for), перехода(break, continue, return, goto)

*45.Условные операторы*  
В языке существует два условных оператора, реализующих ветвление программы:1) оператор if, содержащий проверку одного условия; 2) оператор switch, содержащий проверку нескольких условий.

*46.Оператор if*  
3 способа задания: 1) if (усл.) оператор; 2) if (усл.) { оператор }else оператор }3) if м.б. вложенным. Если рез-т вып. усл. истина то вып. оператор 1, если нет то 2.

*47.Оператор выбора switch*  
Выбор 1 варианта из многих. switch (выражение) {case const1: оператор1, break) case const2: оператор2, break)..defolt (оператор N, break); } Св-ва switch: 1) выр-е должно иметь целочисленный тип 2) defolt м.б. записан в люб. месте. 3) break передает упр. за пределы оператора switch 4) switch отлич. от if тем что может вып. только операции строгого равенства.

*48.Операторы цикла*  
Цикл - многократно повт. участок выполняемого процесса. Каждый переход по телу цикла наз. операцией. Цикл не сод. внутри себя др. циклов наз. простым, содержащий – сложным. Анализируемая переменная наз. параметром цикла. Цикл. процесс- разветвл. пр-сс с 2-мя ветвями из кот. 1 возвр. на предыд. блоки.Кол-во повт. цикла может указываться а м.б. вычислено-это циклы со счетчиком

*49.Оператор while*  
while (усл.) оператор { оператор1…оператор N } Повторн. вып. 1 или неск. операторов в {}, до тех пор пока логич. усл. не примет зн. ложь. While-цикл с предусл., т.к. истинность улс. пров. перед вхождением в цикл, зн. возможна ситуация, когда цикл не вып. Ни разу. В if усл-ю пров. 1 раз, в while-много раз. В while надо вкл. конструкции изменяющие величину проверяемого выр-я, чтобы в итоге оно стало ложным, иначе цикл будет бесконечным

*50.Оператор do…while*do {<оператор>} while (<условие>); Вып. цикла 1 раз, do…while – цикл с постусловием

ш  
for (инициализация; условие; изменение параметров) – цикл с предусловием, может не вып. ниразу. 1) вып.инициализация 2) проверка условия 3) вып. тело цикла 4) вып. изм. парам-в 5) возвр. к пункту 2.

*52.Выбор оператора цикла*

*53.Операторы переходов break, continue, return, goto*  
break-прекращ. вып. цикла и передает выполнение оператору следующему после данного цикла. Continue-пропускает часть цикла стоящую после его записи. Return – возвращ. знач. нек. выр-я, кот. становится значением ф-ии. Goto указывает что вып. пр-ммы необх. продолжить с инструкцией, перед кот записана метка. goto <метка>. В пр-мме должна быть указана строка с меткой, поставлено «:» и записана инструкция к кот. осущ. переход. метка: <идентификатор>

*54.Пустой оператор*  
« ; » при вып. ничего не происходит. Исп. в операторах do, while, for, if когда место оператора не требуется, но по синтаксису требуется

*55.Оператор выражение*  
Люб. выр. кот. заканч. на « ; » явл. оператором. Вызвать ф-ю не возвращ. значения можно только при помощи оператора выражения. a(x,y);

*56.Массивы и указатели в Си*  
Составной тип данных, представляющий собой конечную совокупность элементов одного типа, расположенных в едином блоке памяти строго друг за другом. Размер массива называют количество элементов массива. Обращение к элементам массива производится посредством индекса. Количество индексов, необходимое для обращения к одному элементу, определяет количество измерений массива, то есть размерности

*57.Одномерные массивы*  
Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива. Фактически одномерный массив — это массив, у которого может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива. Обращение к элементам массива: A[i]

*58.Многомерные массивы*  
Многомерные массивы — это массивы, у которых есть более одного индекса. Вместо одной строки элементов, многомерные массивы можно рассматривать как совокупность элементов, которые распределены по двум или более измерениям. Вот так, например, можно визуализировать двумерный массив. Обращение к элементам массива: A[i][j]

*59.Указатели*  
Указатель – вид переменной, кот. хранит адрес эл-в в памяти, где м.б. записано зн. др. переменной. тип \*имя\_указателя; int var, \*point. Операции с указателями: 1) доступа: \*E 2) присваивание: аналогично как и для типов данных. 3) увеличения либо уменьшения указателя: int i, \*p; p+i //указ. на адрес i-го эл-та после данного p-i //перед данным. 4)сложного присваивания: p+=i; p-=i; 5) инкремента и декремента: p++; ++p; p--; --p; 6) индексирования p[i] аналогична \*(E+i), т.е. из памяти извл. и исп. зн. i-го эл-та, адрес кот. присвоен указателю E. 7) вычитание указателей: E1-E2, где E1, E2 – переменные типа указатель на 1 и тот же тип данных, рез-т имеет тип int. 8) отношений: E1==E2; E1>=E2; E1!=E2 и т.д. рез-т имеет тип int

*60.Строки*  
Строка – последоват. люб. символов заключенных в “ ”. Строка предст. со. массив с эл-ми типа char в конце кот. помещен «\о». такой массив наз. строкой формата ASCIIZ. char str[3]={“H”,”I”,\o}; char str[3]=”hi”; char str []=”hi”; scanf(“%s”, str);

*61.Стандартная библиотека работы со строками*  
#include <string.h>; strcopy(char\*dst, char\*src) //копирование строки адресованной src в область памяти на кот. указано dst. size\_t – целочислен. тип исп. для хранен. индексов массивов. strncpy (char\*dst, size\_t n); //копир. только 1-е n-символов в строке src. strdup (char\*s); //выдел. память и копир. в нее содержим. строки s. strlwr(char\*s); //преобраз. все символы строки к нижнему или верхнему регистру. strref(char\*s); //меняет порядок следования символов в строке на противополож-ный. strcut(char\*s); //присоедин. строку src в dst. strcmp(const char\*s1, const char\*s2); //сравнивает строки заданные указателями s1,s2 в лексикограф. порядке, возвр 0 если обе идентичны. strleng (char\*s); //возвр. длину s в байтах. strstr (const char \*s1, const char\*s2); //находит место 11-го вхождения s2 в s1 и возвр. указатель на соотв. позицию

*63.Организация последовательного поиска*  
Последовательный (линейный) поиск – это простейший вид поиска заданного элемента на некотором множестве, осуществляемый путем последовательного сравнения очередного рассматриваемого значения с искомым до тех пор, пока эти значения не совпадут. Идея этого метода заключается в следующем. Множество элементов просматривается последовательно в некотором порядке, гарантирующем, что будут просмотрены все элементы множества (например, слева направо). Если в ходе просмотра множества будет найден искомый элемент, просмотр прекращается с положительным результатом; если же будет просмотрено все множество, а элемент не будет найден, алгоритм должен выдать отрицательный результат.

*62.Поиск и сортировка*  
Поиск — обработка некоторого множества данных с целью выявления подмножества данных, соответствующего критериям поиска. Все алгоритмы поиска делятся на поиск в неупорядоченном и в упорядоченном множестве данных. Упорядоченность – наличие отсортированного ключевого поля. Сортировка - упорядочение (перестановка) элементов в подмножестве данных по какому-либо критерию. Чаще всего в качестве критерия используется некоторое числовое поле, называемое ключевым. Упорядочение элементов по ключевому полю предполагает, что ключевое поле каждого следующего элемента не больше предыдущего (сортировка по убыванию). Если ключевое поле каждого последующего элемента не меньше предыдущего, то говорят о сортировке по возрастанию. Цель сортировки — облегчить последующий поиск элементов в отсортированном множестве при обработке данных.

*65.Основные алгоритмы сортировки массивов*  
Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

*64.Организация бинарного поиска*  
Бинарный (двоичный, дихотомический) поиск – это поиск заданного элемента на упорядоченном множестве, осуществляемый путем неоднократного деления этого множества на две части таким образом, что искомый элемент попадает в одну из этих частей. Поиск заканчивается при совпадении искомого элемента с элементом, который является границей между частями множества или при отсутствии искомого элемента. Бинарный поиск применяется к отсортированным множествам и заключается в последовательном разбиении множества пополам и поиска элемента только в одной половине на каждой итерации. Таким образом, идея этого метода заключается в следующем. Поиск нужного значения среди элементов упорядоченного массива (по возрастанию или по убыванию) начинается с определения значения центрального элемента этого массива. Значение данного элемента сравнивается с искомым значением и в зависимости от результатов сравнения предпринимаются определенные действия. Если искомое и центральное значения оказываются равны, то поиск завершается успешно. Если искомое значение меньше центрального или больше, то формируется массив, состоящий из элементов, находящихся слева или справа от центрального соответственно. Затем поиск повторяется в новом массиве

*66.Сортировка методом пузырька*  
Программа выполнит последовательно следующие действия: (1) Установит размер массива, прося пользователя ввести числовое значение; 2) Заполнит массив значениями, введенными пользователем для каждого элемента массива; 3) Выведет исходный массив; 4) Отсортирует массив по убыванию; 5) Выведет отсортированный массив for(int i = 1; i < n; ++i) { for(int r = 0; r < n-i; r++) { if(mass[r] < mass[r+1]) { // Обмен местами int temp = mass[r]; mass[r] = mass[r+1]; mass[r+1] = temp; } } })

*67.Сортировка методом выбора*  
Это наиболее естественный алгоритм упорядочивания. При данной сортировке из массива выбирается элемент с наименьшим значением и обменивается с первым элементом. Затем из оставшихся n - 1 элементов снова выбирается элемент с наименьшим ключом и обменивается со вторым элементом, и т.д. Шаги алгоритма: 1) находим минимальное значение в текущей части массива; 2) производим обмен этого значения со значением на первой неотсортированной позиции; 3) далее сортируем хвост массива, исключив из рассмотрения уже отсортированные элементы.

*71.Структуры в Си*  
Структура – конструкция, кот. позвол. объедин. несколько эл-в с разными типами и именами в 1 составной объект. Отличия от массива: 1) Могут содерж. данные различн. типов 2) В массиве обращ. к данным идет по номеру, в стр-ре по имени

*68.Сортировка вставками*  
Сортировка вставками (Insertion Sort) — это простой алгоритм сортировки. Суть его заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Стоит отметить что массив из 1-го элемента считается отсортированным. Словесное описание алгоритма звучит довольно сложно, но на деле это самая простая в реализации сортировка. Каждый из нас, не зависимо от рода деятельности, применял алгоритм сортировки, просто не осознавал это:) Например когда сортировали купюры в кошельке — берем 100 рублей и смотрим — идут 10, 50 и 500 рублёвые купюры. Вот как раз между 50 и 500 и вставляем нашу сотню:) Или приведу пример из всех книжек — игра в карточного «Дурака». Когда мы тянем карту из колоды, смотрим на наши разложенные по возрастанию карты и в зависимости от достоинства вытянутой карты помещаем карту в соответствующее место. Для большей наглядности приведу анимацию из википедии.

*69.Быстрая сортировка*  
Общая схема: 1) Есть нек. массив A из n эл-в. Из него выбир. нек. опорный эл-т – p. 2) Запускается процедура разделения массива, кот. перемещ. все эл-ты, кот. ≤p влево от него, а > вправо. 3) Так. обр. мы получили новый амссив из 2-х подмассивов, причем левый < правого. 4) Для обоих подмассивов, если в подмассиве > 2х эл-в, рекурсивно запускаем ту же процедуру. 5) Получаем полностью отсортир. посл. Достоинства: 1) Самый быстродействующий 2) Простая реализация 3) Хорошо рботает на почти отсортированных данных.

*70.Функции пользователя в Си*  
Вып. пр-мы нач. с ф-и main(). В Си запрещено объявлять 1 ф-ю внутри другой. Причины исп. ф-й: 1) Уменьш. дублирования кода 2) Сделать пр-му более структурированной и понятной. Ф-и необх. объявл. в начале пр-мы, такое объявл. наз прототипом ф-ии. Прототип должен предшествовать исполнению ф-ии а полное ее описание м.б. помещено как после так и до тела пр-мы. тип имя ф-ии (тип имя арг.1, арг.2..) Тип перед именем ф-и опр. тип значения, кот. возвр. ф-я. Если тип не указан, то возвр. тип int. Полное опис. ф-й и параметров надо для того чтобы ф-я могла к ним обращаться. int function (int a, int b) //прототип; int function (int a,b,c); Передача знач. из вызванной ф-и в вызвавшею происх. с помощью оператора возврата return\_значение. Вызваная ф-я не мож. менять зн. аргумента вызвавшей ее пр-мы. Но это можно сделать если передавать ф-ям не переменные а их адреса. int swap (int \*a, int \*b); | { int \*tmp; | \*tmp=\*a; | \*a=\*b; | } swap (&b, &c) //передаются адреса b и c, выполнение ф-и приведет к тому, что b и с поменяются местами.

.

*72.Объявление шаблонов структур*  
Общий синтаксис объявления шаблонов стр-р: struct имя шаблона { | тип эл-та: имя 1 | тип эл-та: имя 2 } |. Struct Database { char FAM[20]; | long Tel; | double w; }|. Задание шаблона не влечет резервирование в памяти. struct date { | int day, month, year } | struct FAM[30], inia[15], otch[20]; | struct date Birthday; }

*73.Объявление структур-переменных*  
Ничем не отл. от обычной переменной с предопределенным типом. Синтаксис: struct имя\_шаблона | имя\_переменной. struct date date1[15] //объявл. массив из 15 стр-р | struct person bstu [20]; | \*per\_ptr=&bstu[0]; //объявл. массив из 20 стр-р типа person и задет указатель на данный тип. разрешается совмещать описание шаблона и определение стр-го эл-та. struct date { | int date, month, year } | a,b,date[15], \*ptr\_a=&a;

*74.Доступ к компонентам структуры*  
Осуществл. с пом. оператора «.» при работе со структ. или «->» при исп. указателя на стр-ру. Синтаксис: имя перемен. стр-ры.имя поля; имя указателя->имя поля. Доступ к эл-ту: date1[15].day=10 | struct (bstu[10].FAM, “Иванов”); Доступ по указателю: strcpy((per\_ptr+10)->FAM, “Иванов”); | (per\_ptr+10)->heigh=180

*75.Анонимное определение структуры и оператор определения типа typedef*  
Исп. когда имя стр-ры после ключевого слова struct опускается. В этом случае список опис. перемен. должен быть не пустым. struct { double x; | double y; } | R2point, \*R2point ptr; //опрделены 2 пер.-R2point, R2point ptr. 1-я имеет структ. тип с полями х и у типа double, 2-я указатель на данный структ. тип. Чаще исп. оператор определения типа typedef. typedef struct { | double x; double y; }R2point, \*R2point ptr;

*76.Работа с памятью в Си. Статическая память*  
Для работы с массивами информации, программы должны выделять память для этих массивов. Для выделения памяти под массивы переменных используются соответствующие операторы, функции и т.п*..* Статическое*(*фиксированное*)* выделение памяти. В этом случае память выделяется только один раз во время компиляции. Размер выделенной памяти есть фиксированным и неизменным до конца выполнения программы. Примером такого выделения может служить объявление массива из 10 целых чисел

*77.Работа с памятью в Си. Стековая или локальная память*  
Стек — это область оперативной памяти, которая создаётся для каждого потока. Он работает в порядке LIFO (Last In, First Out),  то есть последний добавленный в стек кусок памяти будет первым в очереди на вывод из стека. Каждый раз, когда функция объявляет новую переменную, она добавляется в стек, а когда эта переменная пропадает из области видимости (например, когда функция заканчивается), она автоматически удаляется из стека. Когда стековая переменная освобождается, эта область памяти становится доступной для других стековых переменных.

*78.Работа с памятью в Си. Динамическая память или куча*  
Куча — это хранилище памяти, также расположенное в ОЗУ, которое допускает динамическое выделение памяти и не работает по принципу стека: это просто склад для ваших переменных. Когда вы выделяете в куче участок памяти для хранения переменной, к ней можно обратиться не только в потоке, но и во всем приложении. Именно так определяются глобальные переменные. По завершении приложения все выделенные участки памяти освобождаются. Размер кучи задаётся при запуске приложения, но, в отличие от стека, он ограничен лишь физически, и это позволяет создавать динамические переменные.

*79.Работа с памятью в Си. Задача: печать n первых простых чисел*

*80.Консольный ввод\вывод. Функции консольного ввода*   
По умолчанию язык C++ не содержит встроенных средств для ввода с консоли и вывода на консоль, эти средства предоставляются библиотекой iostream. В ней определены два типа: istream и ostream. istream представляет поток ввода, а ostream - поток вывода. Для записи или вывода символов на консоль применяется объект cout, который представляет тип ostream. А для чтения с консоли используется объект cin. Для вывода на консоль применяется оператор <<. Этот оператор получает два операнда. Левый операнд представляет объект типа ostream, в данном случае объект cout. А правый операнд - значение, которое надо вывести на консоль.