**1. Структура программы на языке С++**

Все программы на языке C++ содержат директивы препроцессора, описания и функции. Программа должна обрабатывать информацию, значит нужны средства ввода/вывода информации. Для некоторых программ не нужно ввода, но вывод должен быть обязательно.

Препроцессор - специальная программа, которая распознает текст программы. Директива препроцессора - это команда препроцессора языка C++, который автоматически вызывается на первом шаге компиляции программы. Наиболее распространенная директива - #include, которая включает в программу текст внешнего файла.

*#include …;*

После вызова директивы обязательно указать название подключаемого пакета. В отдельных пакетах собраны готовые части, содержащие служебные тексты.

Описания устанавливают имена и типы именованных констант, переменных, функций, а также имена типов, созданных пользователем. Глобальные величины описываются вне операторных скобок любых функций и действуют от конца описания до конца файла. Локальные величины описываются внутри блока и действуют от конца описания до конца блока.

Функция - это набор описаний, выражений и операторов, который выполняет определенную задачу. Функция задается типом, названием и списком параметров, указываемых в круглых скобках (параметров может не быть, но скобки должны быть обязательно!). В зависимости от типа функции, определяется нужно ли выдавать результат. Тело функции заключено в фигурные (операторные) скобки. В языке C++ функции не могут быть вложены друг в друга. Все программы на языке C++ начинают свое выполнение с первого оператора функции main. Фигурные скобки, в которые заключено тело этой функции, определяют начало и конец программы.

**2. Методы доступа к функциям из библиотек и пакетов**

Чтобы воспользоваться функцией из библиотеки необходимо подключить файл, в котором она находится, написав директиву препроцессора #include и далее имя файла в <>, если он находится в стандартных каталогах, в “”, если он находится в каталоге программы. Второй способ также используется для подключения файлов из совсем не стандартных мест, тогда в кавычках указывается полный путь до файла. После этого компилятору необходимо указать, о необходимости использования стандартного пространства имен - using namespace std;. С этого момента все функции, из подключенного файла, могут быть использованы в программе наравне с функциями, описанными в ней самой.

**3. Описания, выражения и операторы в языке С++**

Переменная определяется как пара «имя»-«значение». Имени соответствует ссылка на участок памяти, выделенный переменной, а значением является содержимое этого участка. Именем служит идентификатор, а значение соответствует типу переменной. Переменные типизируются с помощью определений и описаний.

Описания устанавливают имена и типы именованных констант, переменных, функций, а также имена типов, созданных пользователем, используемые в программе. Глобальные величины описываются вне операторных скобок любых функций и действуют от конца описания до конца файла. Локальные величины описываются внутри блока и действуют от конца описания до конца блока.

Все взаимосвязанные атрибуты объектов (тип, класс памяти, области действия имени) приписываются объекту с помощью определений и описаний. Каждое описание делает указанные в нём идентификаторы известными компилятору. Описания сообщают свойства того объекта, к которому относится имя. Описание ассоциирует тип с именем, а определение задает все другие свойства объекта и выполняет его инициализацию. В большинстве случаев определения и описания совпадают.

Выражения - это комбинация операндов, знаков операций и круглых скобок (необходимы для изменения порядка выполнения операций), использующееся для вычисления значения определенного типа. Каждый из операндов, в свою очередь, является выражением, переменной или константой.

Оператор - это наименьшая исполняемая единица программы. Различают операторы выражения, действие которых состоит в вычислении заданных выражений, операторы объявления, составные операторы, пустые операторы, операторы метки, цикла. Для обозначения конца оператора в языке C++ символ точка с запятой. Составной оператор (блок) представляет собой некоторое количество последовательно выполняемых операторов, размещенных между открывающей и закрывающей фигурными (операторными) скобками. За составным оператором точка с запятой не требуется.

**4. Арифметические типы данных в языке С++**

Арифметические типы данных - это типы данных, над которыми определены арифметические операции. Они разделены на две группы: целые и вещественные. Целые: int - целые числа, char - символы, bool - логические значения. Вещественные: float и double. Для типов int и double определены спецификаторы размера long и short. При этом справедливы следующие двойные неравенства:  и , а их конкретные мощности зависят от компилятора. Для целочисленных типов также определены спецификаторы знака signed и unsigned. Спецификатор unsigned позволяет увеличить в 2 раза мощность типа за счет знакового разряда. При использовании в выражениях совместно различных арифметических типов происходит автоматическое приведение к наиболее мощному из них.

-32767<a<32767- *int*- тип целочисленных переменных.

*Long* преобразует вычисляемое вещественное значение к целому «длинному» типа(в результате значение без дробной части, дробная часть отбрасывается при приведении типа).

Действительные числа размером до 10^32 образуют скалярный тип данных - *Float*. Спецификатор типа *float* указывает, что это вещественная арифметическая переменная с внутренним представлением в формате с плавающей точкой. Для десятичных чисел 3-4 знака после запятой – это инженерная точность, однако иногда этого недостаточно, поэтому, чтобы считать с большей точностью, используют тип *Double.* Работа с числами этого типа занимает больше места в памяти*,* следовательно, результат дольше обрабатывается. При этом последняя цифра может быть ошибочной, т.к. используется двоичная система отсчета.

**5. Символьные и логические данные в языке С++**

Величины символьного типа char, как и другие целые типы, могут быть со знаком и без. В величинах со знаком можно хранить значения в диапазоне от –128 до 127. При использовании спецификатора unsigned значения могут находиться в пределах от 0 до 255. Они применяются для хранения символов из 256-символьного набора ASCII. Кроме этого величины типа char применяются также для хранения целых чисел, не превышающих границы указанных диапазонов.

Величины логического типа bool могут принимать только значения true и false, являющиеся зарезервированными словами. Выражения логического типа используются как условия. Внутренняя форма представления значения false — 0 (нуль). Любое другое значение интерпретируется как true. При преобразовании к целому типу true имеет значение 1. При выводе на экран true выводится как 1, а false как 0. Чтобы избежать этого, используется манипулятор вывода: *cout << boolalpha << <логическое\_выражение>;*, тогда на экран будет выводиться true вместо 1 и false вместо 0.

**6. Операции над данными стандартных типов и их старшинство**

В сложных выражениях, содержащих несколько операций, последовательность их выполнения определяется приоритетом операций. Если в выражении встречаются подряд операции одного уровня старшинства, то последовательность их выполнения определяется ассоциативностью. Изменить приоритет выполнения операций можно посредством круглых скобок, их приоритет абсолютен.

**7. Потоковый ввод и вывод информации в языке С++**

В соответствии с названием заголовочного файла iostream.h (stream - поток; "i" - сокращение от input - ввод; "o" - сокращение от output - вывод) описанные в этом файле средства ввода-вывода обеспечивают программиста механизмами для извлечения данных из потоков и для включения (внесения) данных в потоки. Поток определяется как последовательность байтов (символов) и с точки зрения программы не зависит от тех конкретных устройств (файл на диске, принтер, клавиатура, дисплей, стример и т.п.), с которыми ведется обмен данными.

Используемые в программах потоки логически делятся на три типа:

• входные, из которых читается информация;

• выходные, в которые вводятся данные;

• двунаправленные, допускающие как чтение, так и запись.

Все потоки библиотеки ввода-вывода последовательные, т.е. в каждый момент для потока определены позиции записи и (или) чтения, и эти позиции после обмена перемещаются по потоку на длину переданной порции данных.

Потоковый ввод и вывод становятся доступны пользователю после подключения файла iostream. С этого момента появляется доступ к объектам cin - консольный ввод и cout - консольный вывод. С этими объектами связаны две операции, полученные путем перегрузки стандартных операций поразрядных сдвигов: для cin - >> извлечение из потока, для cout - << помещение в поток. Эти операции перегружены для всех встроенных типов данных, что позволяет автоматически выполнять ввод и вывод в соответствии с типом величин.

**8. Манипуляторы потокового ввода и вывода**

Манипуляторами называют специальные функции, позволяющие программисту изменять состояния и флаги потока. Особенность манипуляторов и их отличие от обычных функций состоит в том, что их имена (без параметров) и вызовы (с параметрами) можно использовать в качестве правого операнда для операции обмена (<< или >>). В качестве левого операнда в этом выражении, как обычно, используется поток (ссылка на поток), и именно на этот поток оказывает влияние манипулятор.

Манипуляторы библиотеки классов ввода-вывода языка С++ делятся на две группы: манипуляторы с параметрами (содержатся в файле iomanip) и манипуляторы без параметров (содержатся в файле iostream).

***endl*** - действует только при выводе, включает в выходной поток символа новой строки и сбрасывает буфер этого потока;

***boolalpha*** и ***noboolalpha -*** вывод логических величин словами или цифрами;

***showpoint*** и ***noshowpoint -*** вывод действительных величин с точкой или без, но только, если отсутствует дробная часть;

***showpos*** и ***noshowpos -*** вывод целочисленных и действительных положительных величин со знаком «+» или без;

***left***, ***internal*** и ***right -*** выравнивание по левому краю, по середине, по правому краю;

***fixed*** и ***scientific*** - вывод действительных величин в формате с фиксированной и плавающей точкой.

***setprecision(int) -*** количество разрядов после запятой действительной величины;

***setw(int)*** - ширина контейнера вывода.

**9. Ссылочные типы данных и их применение**

Ссылка представляет собой новое имя величины, указанной при ее инициализации. Ссылаться можно на любой тип данных как стандартный так и определенный пользователем. Для создания объекта ссылочного типа достаточно после указания типа добавить символ &. Ссылка должна быть инициализирована при описании. После инициализации ссылке не может быть присвоена другая переменная. Тип ссылки должен совпадать с типом величины, на которую она ссылается. Не допустимы указатели на ссылки, массивы ссылок и ссылки на ссылки.

Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений, что позволяет функции работать с переменной напрямую, не занимая дополнительной памяти.

**10. Указатели в языке С++. Доступ к данным по указателю**

Указатель - это переменная, содержащая адрес другой переменной, т.е. значением переменной типа указатель является целое число, равное адресу того объекта, на который ссылается указатель. Для создания указателя достаточно перед именем добавить символ \*. Указатель можно создать на любой тип как стандартный так и определенный пользователем за исключением ссылочных. В C++ различают указатели на объект и на функцию. Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа, в том числе возможен указатель на указатель <тип> \*<имя\_указателя> = &<имя>;. Среди указателей на объект можно выделить указатель на тип void. Он применяется а тех случаях, когда конкретный тип объекта, на который необходим указатель, не определен. Перед выполнением каких-либо действий с областью памяти, на которую он указывает, требуется преобразовать его к конкретному типу. После этого с ним можно работать как с указателем этого конкретного типа. Указатель на функцию содержит адрес, по которому передается управление при вызове функции <тип> (\*<имя>) (<список\_аргументов\_функции>);.

**11. Указатели в языке С++. Связь указателей, адресов и типов данных**

Указатели предназначены для хранения адресов областей памяти. Для создания указателя достаточно перед именем добавить символ \*. Указатель можно создать на любой тип как стандартный так и определенный пользователем за исключением ссылочных. В C++ различают указатели на объект и на функцию. Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа, в том числе возможен указатель на указатель <тип> \*<имя\_указателя> = &<имя>;. Среди указателей на объект можно выделить указатель на тип void. Он применяется а тех случаях, когда конкретный тип объекта, на который необходим указатель, не определен. Перед выполнением каких-либо действий с областью памяти, на которую он указывает, требуется преобразовать его к конкретному типу. После этого с ним можно работать как с указателем этого конкретного типа. Указатель на функцию содержит адрес, по которому передается управление при вызове функции <тип> (\*<имя>) (<список\_аргументов\_функции>);.

Указатель существенно связан с типом объекта, на который он ссылается. Если в описании перед обозначением объекта поставить символ "\*", то оно будет описывать указатель на объект того же типа и класса памяти, которые соответствуют данному обозначению без звездочки. Значением, хранимым указателем, является адрес в оперативной памяти, но по этому адресу можно хранить только данные того же типа, что и указатель.

**12. Программная реализация алгоритмов линейной структуры**

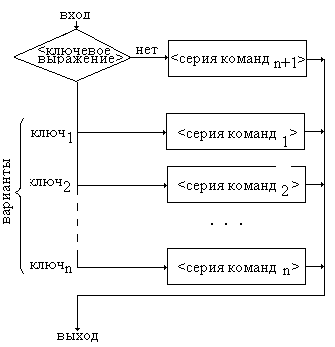
Алгоритмы линейной структуры описывают линейный вычислительный процесс, этапы которого выполняются однократно и последовательно один за другим. Программную реализацию такой алгоритмической структуры называют линейной программой. Линейные программы обычно предназначены для решения простейших задач. Такие программы представляют собой последовательность операторов, каждый из которых не предполагает ветвления, передачи управления в другое место и циклического выполнения. Алгоритм программы с линейной структурой может быть представлен следующим образом:

**13. Программная реализация разветвляющихся алгоритмов**

Разветвляющийся алгоритм описывает вычислительный процесс, реализация которого происходит по одному из заранее предусмотренных направлений. Разветвляющийся алгоритм содержит блок проверки условия. В зависимости от результата проверки условия выполняется та или иная последовательность операций. Направления, по которым может следовать вычислительный процесс, называются ветвями. При этом форма разветвления может быть полной или сокращенной.

Полная форма разветвления:

Сокращенная форма разветвления:

****

Второй алгоритмической структурой, определяющей разветвляющийся алгоритм, является структура выбора.

Структура выбора используется в алгоритмах, в которых при разных значениях одного и того же выражения (которое называют ключевым выражением) необходимо выполнять различные действия (различные варианты действий, каждый из которых имеет свой уникальный ключ варианта). Выполняется тот вариант, к которому "подошел ключ", полученный в результате вычисления ключевого выражения (значение ключевого выражения и константа, представляющая ключ варианта совпали).

Для программной реализации разветвляющихся алгоритмов в языке С++ используются операторы выбора - это условный оператор (if) и переключатель (switch).

Рассмотрим оператор if.

**if (<выражение>) <оператор;> [ else <оператор;>]**

где в качестве <выражения> могут использоваться арифметические выражения, отношение и логическое выражение .

Если результат выражения отличен от 0 (**true**), то выполняется оператор, следующий за скобкой, иначе (**false**) - выполняется оператор, следующий за служебным словом **else**.

Переключатель switch.

Оператор выбора используется при программировании выбора одной из многих ветвей.

**switch(<выражение>){**

**case<элемент>: <операторы;>**

**case<элемент>: <операторы;>**

**. . .**

**[ default : <операторы;>] }**

Результат выражения сравнивается с заданными значениями, и, в случае равенства, выполняются соответствующие операторы, которых может быть 0 или более. Затем выполняются операторы всех последующих альтернатив, если не встретится оператор **break**.

**14. Структурный тип данных**

Структура, в отличие от массива и строки, может содержать элементы различных типов. Элементы структуры называются полями структуры и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Этим достигается создание связных списков структур.

Описание структур может выполняться двумя способами.

**struct [<имя структуры>] {<описание полей>} [<список переменных>];**

Второй способ описания структур осуществляется с помощью оператора объявления нового типа **typedef.** С использованием этого оператора создается новый тип данных – структура

**typedef struct {<описание полей>} [<имя структуры>];**

**[<имя структуры>] [<список переменных>];**

Обращение к элементам структуры выполняется с указанием имени переменной и имени поля:

**<имя переменной>.<имя поля>**

Структуры при объявлении можно сразу инициализировать. При этом значения полей указываются после знака равенства в фигурных скобках через запятую:

**[<имя структуры>] [<имя переменной типа структура>] = {<значения полей>};**

Возможно объявление массивов структур и структур, элементами которых являются вложенные структуры.

**15. Массивы в языке С++. Связь массивов и указателей**

Массив - конечная последовательность элементов одного и того же типа. Каждый элемент массива имеет селектор (индекс, номер). С его помощью можно определить любой элемент. Описание и инициализация одномерного массива **<тип> <имя>[<целочисленная\_константа>] = {<значение1>, <значение2>, …};**.Константа в квадратных скобка представляет собой размерность массива.

В С++ индексы всегда начинаются с 0, поэтому индекс последнего элемента массива на единицу меньше размера массива.

Имя массива система понимает как указатель на этот массив. Поэтому чтобы получить доступ к элементу массива, также можно **\*(<имя\_массива>+<номер\_элемента\_массива>);**. Так как C++ передает массив в функцию с помощью адреса массива функция может изменять значения, содержащиеся в массиве.

Можно объявить массив, не указывая явно, сколько в нем элементов. При этом система должна знать, сколько места в памяти будет этот массив занимать. Каждому массиву при обработке нужно задать еще один параметр: сколько элементов в нем в действительности находится. Исключение: массивы символов. Каждый символ занимает 1 байт. Символьные массивы отличаются от всех остальных тем, что после символов стоит значок - символ, означающий, что перечисление символов закончено. Поэтому в памяти выделяется на 1 байт больше содержащихся значений.

Допускается создание многомерных массивов как массивов, элементами которых являются массивы.

**16. Многомерные массивы**

Многомерные массивы в C++ рассматриваются как массивы, элементами которых являются массивы. Определение многомерного массива в общем случае должно содержать сведения о типе, размерности и количествах элементов каждой размерности. В памяти массив ARRAY размещается в порядке возрастания самого правого индекса, то есть самый младший адрес имеет элемент ARRAY[0][0][0], затем идет элемент ARRAY[0][0][1] и т.д.

Компилятор первым делом вычисляет, сколько всего элементов в массиве умножается на double, отводится место в памяти.

В массиве, где n индексов, при вычислении функции делается n умножений. Поэтому для работы с отдельно взятым элементом массива лучше использовать еще одну ячейку памяти. Чтобы ускорить процесс работы системы нужно не каждый раз вызывать переменную из массива, а, вызвав 1 раз, вложить новую переменную и совершить нужные действия.

С учетом порядка расположения в памяти элементов многомерного массива нужно размещать начальные значения его элементов в списке инициализации.

Если многомерный массив при определении инициализируется, то его самая левая размерность может в скобках не указываться. Количество элементов компилятор определяет по числу членов в инициализирующем списке.

**17. Строки и массивы символов**

Строки и массивы символов используются для хранения упорядоченных последовательностей символов. Массив символов – это одномерный массив типа char. К элементу такого массива как и к элементу любого другого массива можно обратиться по номеру, но операцию присваивания можно проводить сразу ко всему массиву символов, записав справа от знака = не символьную, а строковую константу, а слева не элемент массива, а имя массива. При этом в описании не обязательно задавать размеры массива, поскольку компилятор "сам" определяет его длину, подсчитывая число начальных значений. Массив символов как и строка оканчивается символом ‘\0’. Это нуль-символ: в языке C++ он используется для того, чтобы отмечать конец строки. Нуль-символ - не есть цифра 0; он не выводится на печать и в таблице кодов ASCII имеет номер 0. Наличие нуль-символа означает, что количество ячеек массива должно быть по крайней мере на одну больше, чем количество символов, которые необходимо размещать в памяти. Языку C++ досталось в наследство от языка C отсутствие типа строка. Соответствующий класс можно получить подключив директивой препроцессора #include файл string.h. С этого момента доступен будет не только сам класс строк, но и операции по их обработке, такие как сцепление строк (+), операции посимвольного сравнения (<, <=, >, >=), присваивание (=), полученные перегрузкой операций, а также методы (функции) по работе со строками.

**18. Реализация циклических алгоритмов с заранее неизвестным количеством повторений**

Для программирования циклических процессов с заранее не известным количеством повторений в С++ используются операторы цикла с предусловием **while** и с постусловием **do ... while**.

Цикл с предусловием **while (<логическое\_выражение>) <оператор>;**. Операторы тела цикла будут выполняться до тех пор пока логическое выражение не вернет ложь, при этом если оно будет ложным изначально, то операторы тела цикла ни выполнятся ни разу. Это и есть фундаментальное различие между этими двумя операторами.

Цикл с постусловием **do <оператор> while (<логическое\_выражение>);**. Операторы тела цикла выполнятся 1 раз, а затем будет вычислено логическое выражение, если оно окажется истинно, то тело цикла выполнится снова, если же оно окажется ложно, то выполнение оператора прекратится.

Алгоритмы программ с циклическими алгоритмами могут быть представлены следующим образом:

**19. Реализация циклических алгоритмов с заранее известным количеством повторений**

Для программирования циклических процессов с заранее известным количеством повторений в С++ используется оператор цикла с известным количеством повторений for.

Цикл с известным количеством повторений **for ([<единожды\_выполняемые\_операторы>(1)]; [<логическое\_выражение>(2)]; [<операторы\_выполняемые\_на\_каждой\_итерации\_цикла>](3)) <оператор>;**. Любое из выражений (1) или (2) или (3), а также все сразу, могут быть опущены, при этом разделяющие их символы «;» пропускать нельзя. Операторы тела цикла будут выполняться до тех пор пока логическое выражение не вернет ложь, при этом если оно будет ложным изначально, то операторы тела цикла также как и операторы, выполняемые на каждой итерации цикла, ни выполнятся ни разу и управление передастся на оператор, следующий за оператором **for**. Единожды выполняемые операторы и операторы, выполняемые на каждой итерации цикла, по своему количеству никак не ограничены и между собой разделяются запятой.

Другим вариантом использования оператора **for** является бесконечный цикл. Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют условный оператор и оператор **break**.

**for ( ; ; )**

**{ ...**

**if (<некоторое условие>) break;**

**... }**

Согласно синтаксису языка Си тело цикла должно иметься у любого цикла, но если по смыслу оно не требуется, то в качестве тела можно использовать пустой оператор. Такой цикл может быть использован например для организации поиска.

**20. Структура функции в языке С++. Прототип, заголовок и вызов функции**

**<тип> <имя> ([список\_формальных\_параметров]) {<тело\_функции>}**

Прототип - это явное объявление функции, которое предшествует определению функции. Тип возвращаемого значения при объявлении функции должен соответствовать типу возвращаемого значения в определении функции.

Если прототип функции не задан, а встретился вызов функции, то строится неявный прототип из анализа формы вызова функции. Тип возвращаемого значения создаваемого прототипа int, а список типов и числа параметров функции формируется на основании типов и числа фактических параметров используемых при данном вызове.

Таким образом, прототип функции необходимо задавать в следующих случаях:

1. Функция возвращает значение типа, отличного от int.

2. Требуется проинициализировать некоторый указатель на функцию до того, как эта функция будет определена.

Наличие в прототипе полного списка типов аргументов параметров позволяет выполнить проверку соответствия типов фактических параметров при вызове функции типам формальных параметров, и , если необходимо, выполнить соответствующие преобразования.

В прототипе можно указать, что число параметров функции переменно, или , что функция не имеет параметров.

Заголовок функции задает ее имя, тип возвращаемого значения и список, передаваемых параметров. За заголовком функции следует ее определение. Также заголовок может использоваться как прототип.

Вызов функции имеет следующий формат:

**адресное-выражение ([список-выражений])**

Поскольку синтаксически имя функции является адресом начала тела функции, в качестве обращения к функции может быть использовано адресное-выражение, имеющее значение адреса функции.

Список-выражений представляет собой список фактических параметров, передаваемых в функцию. Этот список может быть и пустым, но наличие круглых скобок обязательно.

Фактический параметр может быть величиной любого основного типа, структурой, объединением, перечислением, или указателем на объект любого типа. Массив и функция не могут быть использованы в качестве фактических параметров, но можно использовать указатели на эти объекты.

Выполнение вызова функции происходит следующим образом:

1. Вычисляются выражения в списке выражений и подвергаются обычным арифметическим преобразованиям. Затем, если известен прототип функции, тип полученного фактического аргумента сравнивается с типом соответствующего формального параметра. Если они не совпадают, то либо производится преобразование типов, либо формируется сообщение об ошибке.

2. Происходит присваивание значений фактических параметров соответствующим формальным параметрам.

3. Управление передается на первый оператор функции.

4. Выполнение оператора return в теле функции возвращает управление и возможно, значение в вызывающую функцию. При отсутствии оператора return управление возвращается после выполнения последнего оператора тела функции, а возвращаемое значение не определено.

Адресное выражение, стоящее перед скобками определяет адрес вызываемой функции. Это значит, что функция может быть вызвана, через указатель на функцию.

Формальные аргументы при описании функции показывают, что будет делаться с фактическими аргументами, когда функция будет вызвана. Для вызова функции достаточно написать ее имя со списком передаваемых фактических аргументов.

Компилятор использует объявление заголовка функции для сравнения типов фактических аргументов при вызове функции с формальными параметрами даже при отсутствии явного описания.

**21. Предварительное описание функции**

Предварительное описание функции задает ее имя, тип возвращаемого значения и список типов, передаваемых параметров. Список прототипов всех функций, как правило, размещается перед определением первой функции и необходим при вызове функциями друг-друга, т.е. функция может вызвать функцию, описание которой расположено после этой функции.

**<тип> <имя> ([список\_типов\_формальных\_параметров]);**

При записи функций в библиотечные файлы, прототипы отправляются в файлы с расширением \*.h.

**22. Локальные и глобальные величины программы**

Глобальные величины описываются вне операторных скобок любых функций и действуют от конца описания до конца файла. Локальные величины описываются внутри блока и действуют от конца описания до конца блока. Блоком является любая пара фигурных скобок. Внутри блока могут находится другие блоки, тогда локальная величина для этих блоков является глобальной, но только, если в этом блоке не определена переменная того же типа и имени.

**23. Способы обмена информацией между вызывающей и вызываемой функциями**

Механизм параметров является основным способом обмена информацией между вызываемой и вызывающей функциями. Параметры, перечисленные в заголовке описания функции, называются формальными, а записанные при вызове функции - фактическими (или аргументами). При вызове функции в первую очередь вычисляются выражения, стоящие на месте фактических параметров; затем выделяется память под формальные параметры функции в соответствии с их типом, и каждому из них присваивается значение соответствующего фактического параметра.

Существует два вида передачи величин в функцию: по значению и по адресу.

При передаче по значению в функцию передаются копии значений фактических параметров, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет и возможности их изменить.

При передаче по адресу в функцию передаются копии адресов параметров, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения параметров.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются. Поэтому использование ссылок вместо указателей улучшает читаемость программы. Использование ссылок вместо передачи по значению более эффективно, поскольку не требует копирования параметров.

Параметры, передаваемые в функцию, могут быть любого типа, кроме массива или функции, которые передаются с помощью указателей.

Второй способ передачи параметров – это использование величин объявленных вне всяких функций. Такие величины видимы для всех функций, описанных после них.

**24. Передача функций в качестве параметров функций**

Функцию можно вызвать через указатель на нее. Для этого объявляется указатель соответствующего типа и ему с помощью операции взятия адреса присваивается адрес функции

**<тип> (\*<имя\_указателя>)(<список\_типов\_формальных\_параметров>) = &<имя\_функции>;**.

Передача функции в другую функцию **<имя\_функции>(<имя\_указателя>[, …]);**.

Вызов функции, переданной через указатель **<имя\_указателя>(<список\_фактических\_параметров>);**.

Тип указателя и тип функции, которая вызывается посредством него, должны совпадать.

**25. Перегрузка функций в языке С++**

Перегрузкой функций называется создание нескольких функций с одним именем, но разными типами параметров или хотя бы различия должны быть в порядке их следования. При этом тип возвращаемого функциями результата не имеет значения. Компилятор определит какую из функций вызывает пользователь по списку фактических параметров. Такое необходимо, когда один и тот же алгоритм необходимо осуществить для различных типов данных.

**26. Шаблоны функций**

С помощью шаблона функции можно определить алгоритм, который будет применяться к данным различных типов, а конкретный тип передается функции в виде параметра на стадии компиляции. Шаблоны функций позволяют объявлять типонезависимые, или общие, функции. Когда программе требуется использовать функцию с определенными типами данных, она должна указать прототип функции, который определяет требуемые типы. Когда компилятор C++ встретит такой прототип функции, он создаст операторы, соответствующие этой функции, подставляя требуемые типы. Программы должны создавать шаблоны для общих функций, которые работают с отличающимися типами. Другими словами, если с какой-либо функцией используется только один тип, нет необходимости применять шаблон. Если функция требует несколько типов, шаблон просто назначает каждому типу уникальный идентификатор, например Т, T1 и Т2. Позже в процессе компиляции компилятор C++ корректно назначит типы, указанные в прототипе функции.

**28. Структура рекурсивных алгоритмов**

Рекурсивной называется функция, которая вызывает сама себя. Такая рекурсия называется прямой. Существует еще косвенная рекурсия, когда две или более функции вызывают друг друга. Если функция вызывает себя, то в памяти создается копия значений ее параметров, после чего управление передается первому исполняемому оператору функции. При каждом следующем вызове данный процесс повторяется. Для выхода необходимо, чтобы функция содержала хотя бы одну не рекурсивную ветвь алгоритма. При завершении функции, управление передается вызывающей функции, выполнение которой продолжается с точки, следующей за рекурсивным вызовом.

Рекурсивные функции применяют для реализации рекурсивных алгоритмов, которые применяют для упрощения реализации, т.к. любой рекурсивный алгоритм можно записать без применения рекурсии.

**27. Оценки временной сложности алгоритма**

Временная сложность алгоритма (в худшем случае) — это функция размера входных и выходных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера. Во многих задачах размер выхода не превосходит или пропорционален размеру входа — в этом случае можно рассматривать временную сложность как функцию размера только входных данных.

Аналогично понятию временной сложности в худшем случае определяется понятие временная сложность алгоритма в наилучшем случае. Также рассматривают понятие среднее время работы алгоритма, то есть [математическое ожидание](http://ru.wikipedia.org/wiki/Математическое_ожидание) времени работы алгоритма.

Для оценки временной эффективности алгоритмов известны вероятностный и статистические подходы. Вероятностный анализ строится на анализе алгоритма (без его реализации), основываясь на вероятностных характеристиках входных данных. Статистический анализ — на основании результатов многократного выполнения реализованного на ЭВМ алгоритма статистическими методами.

Вероятностный анализ де-факто стал общепризнанным. Основным показателем временной эффективности считается временная сложность (ВС) или асимптотическая временная сложность алгоритмов. Справедливо считается, что статистическая оценка имеет погрешности связанные с архитектурой компьютера, компилятора, квалификацией программиста, выбором для исследования множества входных данных.

Однако, вероятностный подход также имеет ряд недостатков, которые рассмотрим на примере алгоритмов сортировки. Во-первых, ВС их в значительной мере зависит не только от объема, но и самих данных, их типа. Для сортировки это выражается в степени первоначальной упорядоченности данных. При сортировке простым включением она изменяется от O(n) до O(n2). Во-вторых, при различных условиях сравнение ВС двух алгоритмов могут давать противоположные результаты в зависимости от неизвестных или часто трудноопределимых коэффициентов пропорциональности. Для среднестатистического случая количество пересылок при n<5 в сортировке простым обменом меньше, чем в сортировке простым включением, а при n>5 наоборот больше, а ВС у них одинакова. В-третьих, — этот показатель не является достаточным для прогноза времени выполнения программ и, как следствие, ограничен в использовании при выборе алгоритма из нескольких конкурентоспособных. В показано, что реализации двух различных алгоритмов сортировки при одних и тех же данных на ЭВМ одной линии, но с различной архитектурой могут изменять факт превосходства одного над другим.

Предлагается статистическая оценка временной эффективности алгоритмов. Определяется область действия показателей, которая включает:

множество возможных значений объемов данных;

множество возможных типов данных;

множество возможных значений данных;

множество трансляторов (раздельно с и без оптимизации);

множество возможных архитектур ЭВМ, на которых возможна реализация алгоритмов.

Для вычисления оценок предлагается метод Монте-Карло. Проводятся измерения времени выполнения программ со случайным выбором значений из перечисленных выше пяти множеств.

В качестве статистических оценок предлагаются:

степень превосходства одного алгоритма над другим – среднее значение разности времен выполнения к максимальному из них;

область превосходства – среднее количество превосходства в проведенных численных экспериментах;

превосходство в пределах – тенденция изменения степени превосходства при максимальном увеличении объема данных.

**29. Динамические переменные. Выделение и освобождение памяти при выполнении программы**

Динамическое распределение памяти используется для экономии памяти. Те переменные, которые становятся ненужными, уничтожаются, а освобожденное место используется для новых переменных. Это особенно эффективно в задачах, в которых число необходимых объектов зависит от обрабатываемых данных или от действий пользователя, т.е. заранее неизвестно. В таких ситуациях можно с запасом предусмотреть место под максимально возможное количество объектов, но крайне ресурсоемко.

Динамическое распределение памяти осуществляется с помощью операций new и delete. Операция <тип> <имя\_указателя> = new <тип>[(<значение>)]; возвращает указатель на переменную типа, указанного в угловых скобках, при этом можно сразу же разместить по этому адресу некоторое значение, для этого его достаточно записать в круглых скобках после типа. Операция delete <имя\_указателя>; освобождает область памяти, на которую указывал указатель.

**30. Построение списковых (последовательных) структур**

Списковыми называются структуры, среди полей которых присутствует указатель на аналогичную структуру. К каждой структуре можно прийти из одной другой структуры и уйти в одну третью структуру. Подобная конструкция облегчает добавление, удаление и перемещение структур по сравнению с массивом.

Построение списковых структур:

1. Объявляем структуру

**struct <имя\_структуры> {<информационная\_часть\_структуры>; <имя\_структуры> <указатель\_на\_структуру>;};**

1. Объявляем 3 указателя на структуры

**<имя\_структуры> <указатель> = NULL, <указатель\_на\_новую\_структуру>, <указатель\_на\_старую\_структуру>;**

1. Выделяем память под новую структуру

**<указатель\_на\_новую\_структуру> = new <имя\_структуры>;**

1. Заполняем поля новой структуры, указателю присваиваем значение NULL. Помещаем вновь созданную структуру в конец списка

**if (<указатель> == NULL) <указатель> = <указатель\_на\_новую\_структуру>; else <указатель\_на\_старую\_структуру>.<указатель\_на\_структуру> = <указатель\_на\_новую\_структуру>; <указатель\_на\_старую\_структуру> = <указатель\_на\_новую\_структуру>;**

**31. Понятие информации, виды и способы ее представления**

Информация - это отражение предметного мира с помощью знаков и сигналов.

Основные виды информации по ее форме представления, способам ее кодирования и хранения, что имеет наибольшее значение для информатики, это:

**графическая или изобразительная** — первый вид, для которого был реализован способ хранения информации об окружающем мире в виде наскальных рисунков, а позднее в виде картин, фотографий, схем, чертежей на бумаге, холсте, мраморе и др. материалах, изображающих картины реального мира;

**звуковая** — мир вокруг нас полон звуков и задача их хранения и тиражирования была решена с изобретение звукозаписывающих устройств в 1877 г.; ее разновидностью является музыкальная информация — для этого вида был изобретен способ кодирования с использованием специальных символов, что делает возможным хранение ее аналогично графической информации;

**текстовая** — способ кодирования речи человека специальными символами — буквами, причем разные народы имеют разные языки и используют различные наборы букв для отображения речи; особенно большое значение этот способ приобрел после изобретения бумаги и книгопечатания;

**числовая** — количественная мера объектов и их свойств в окружающем мире; особенно большое значение приобрела с развитием торговли, экономики и денежного обмена; аналогично текстовой информации для ее отображения используется метод кодирования специальными символами — цифрами, причем сис-темы кодирования (счисления) могут быть разными;

**видеоинформация** — способ сохранения «живых» картин окружающего мира, появившийся с изобретением кино.

Информация - очень емкое понятие, в которое вмещается весь мир: все разнообразие вещей и явлений, вся история, все тома научных исследований, творения поэтов и прозаиков. И все это отражается в двух формах - непрерывной и дискретной.

Объекты и явления характеризуются значениями физических величин. Природа некоторых величин такова, что величина может принимать принципиально любые значения в каком-то диапазоне. Эти значения могут быть сколь угодно близки друг к другу, исчезающе малоразличимы, но все-таки, хотя бы в принципе, различаться, а количество значений, которое может принимать такая величина, бесконечно велико.

Такие величины называются непрерывными величинами, а информация, которую они несут в себе, непрерывной информацией.

Величины, принимающие не всевозможные, а лишь вполне определенные значения, называют дискретными. Для дискретной величины характерно, что все ее значения можно пронумеровать целыми числами 0,1,2,...

Можно утверждать, что различие между двумя формами информации обусловлено принципиальным различием природы величин. В то же время непрерывная и дискретная информация часто используются совместно для представления сведений об объектах и явлениях.

1, В качестве носителей непрерывной информации могут использоваться любые физические величины, принимающие непрерывный “набор” значений (правильнее было бы сказать принимающие любое значение внутри некоторого интервала). Физические процессы (перемещение, электрический ток и др.) могут существовать сами по себе или использоваться, например, для передачи энергии. Но в ряде случаев эти же процессы применяются в качестве носителей информации. Чтобы отличить одни процессы от других, введено понятие “сигнал”. Если физический процесс, т.е. какая-то присущая ему физическая величина, несет в себе информацию, то говорят, что такой процесс является сигналом. Именно в этом смысле пользуются понятиями “электрический сигнал”, “световой сигнал” и т.д. Таким образом, электрический сигнал - не просто электрический ток, а ток, величина которого несет в себе какую-то информацию.

С помощью чисел можно перенумеровать все “разнообразия” реального мира. Именно такая цифровая форма представления информации используется в ЭВМ. Основной элемент в этой форме - слово. Слово - имя объекта, действия, свойства и т.п., с помощью которого выделяется именуемое понятие в устной речи или в письменной форме.Слова строятся из букв определенного алфавита (например, А, Б, ... , Я). Кроме букв используются специальные символы - знаки препинания, математические символы +, -, знак интеграла, знак суммы и т.п. Все разнообразие используемых символов образует алфавит, на основе которого строятся самые разные объекты:

из цифр - числа;

из букв - собственно слова,

из цифр, букв и математических символов - формулы и т.д.

И все эти объекты несут в себе информацию :

числа - информацию о значениях;

слова - информацию об именах и свойствах объектов;

формулы - информацию о зависимостях между величинами и т.д.

Эта информация (и это очевидно) имеет дискретную природу и представляется в виде последовательности символов. О такой информации говорят как об особом виде дискретной информации и называют этот вид символьной информацией. Наличие разных систем письменности, в том числе таких, как иероглифическое письмо, доказывает, что одна и та же информация может быть представлена на основе самых разных наборов символов и самых разных правил использования символов при построении слов, фраз, текстов. Итак, символьная информация может представляться с использованием самых различных алфавитов (наборов символов) без искажения содержания и смысла информации: при необходимости можно изменять форму представления информации - вместо общепринятого алфавита использовать какой-либо другой, искусственный алфавит, например, двухбуквенный. Форма представления информации, отличная от естественной, общепринятой, называется кодом. Коды широко используются в нашей жизни: почтовые индексы, телеграфный код Морзе и др. Широко применяются коды и в ЭВМ и в аппаратуре передачи данных. Так, например, широко известно понятие “программирование в кодах”. Кроме рассмотренных существуют и другие формы представления дискретной информации.

**32.Задачи получения, передачи, преобразования и хранения информации**

Обработка информации — получение одних информационных объектов из других путем выполнения некоторых алгоритмов.

Обработка является одной из основных операций, выполняемых над информацией, и главным средством увеличения объема и разнообразия информации.

Средства обработки информации — это всевозможные устройства и системы, созданные человечеством, и в первую очередь компьютер — универсальная машина для обработки информации.

Компьютеры обрабатывают информацию путем выполнения некоторых алгоритмов. Живые организмы и растения обрабатывают информацию с помощью своих органов и систем.

Информационные ресурсы — это идеи человечества и указания по реализации этих идей, накопленные в форме, позволяющей их воспроизводство.

Информационные ресурсы (в отличие от всех видов ресурсов — трудовых, энергетических, минеральных ) тем быстрее растут, чем больше их расходуют.

Информационные технологии - это совокупность методов и устройств, используемых людьми для обработки информации.

К способам представления информации можно отнести: всевозможные символы, языки и коды (русский, английский, азбука Морзе, язык жестов, язык мимики), чертежи, картины, музыкальные произведения, кино- и видео фильмы, компьютерные игры и т.п.

Хранить информацию можно:

\* в форме знаков на бумаге, перфокартах, перфоленте, на камне, дереве, ткани, в рельефе ключа и т.п.;

\* в форме электромагнитных сигналов на дисках, дискетах, кассетах,

\* кинопленке и т.п.;

\* в форме структурных комбинаций в клетках и генах биологических объектов;

Передача информации происходит:

\* в форме сигналов, как в технических устройствах, так и в общении людей, в телекоммуникациях, в жизни общества, в живых существах;

\* механическим путем в общении людей и в технических устройствах.

К техническим средствам хранения, передачи и обработки информации относятся:

- различные диски,

- дискеты,

- аудио и видео кассеты,

- перфокарты,

- перфоленты,

- кинопленка,

- объекты телекоммуникации (радио, телевидение, телефон, телеграф, персональные компьютеры и компьютерные сети и т.п.)

ЭВМ - это универсальная вычислительная машина, которая обрабатывает информацию, используя электрические сигналы.

Если сигнал непрерывен во времени, то его называют АНАЛОГОВЫМ, и машины, использующие такие сигналы, называют АНАЛОГОВЫМИ.

Если сигнал имеет прерывистую (дискретную) форму, он называется ЦИФРОВЫМ (дискретным), и машины соответственно называются ЦИФРОВЫМИ. Современные ПК относятся к ЦВМ.

Всю информацию ПК может воспринимать в форме сигналов, которые имеют только два различных состояния: есть сигнал или нет сигнала.

Этим сигналам поставлен в соответствие разряд кода, который может принимать только два значения: 0 или 1.

Сигнал, который имеет только два различных значения, или соответствующий ему разряд кода, который может принимать только два значения - 0 или 1, называется бит. Бит - это минимальная единица информации (англ. Bit- binary digit — двоичная цифра).

На практике чаще применяется более крупная единица — байт, равная восьми битам. Именно восемь бит требуется для того, чтобы закодировать любой из 256 символов (256 = 28).

1 байта достаточно для кодирования одного знака текстовой, числовой, звуковой или графической информации.

Крупные производные единицы информации:

1 Килобайт (Кбайт) = 1024 байт

1 Мегабайт (Мбайт) = 1024 Кбайт

1 Гигабайт (Гбайт) = 1024 Мбайт

1 Терабайт(Тбайт) = 1024Гбайт

1 Петабайт(Пбайт)= 1024 Тбайт

1 Экзабайт (Эбайт)=1024 Пбайт

1 Зеттабайт (Збайт)= 1024 Эбайт

1 Йоттабайт (Йбайт)= 1024 Збайт

**33.Язык как способ представления и передачи информации**

Знаковая форма восприятия, хранения и передачи информации означает использование какого-либо языка. Языки делятся на разговорные (естественные) и формальные. Естественные языки носят национальный характер. Формальные языки чаще всего относятся к специальной области человеческой деятельности (например, язык математики или язык флажков на флоте).

Информация во внешней среде выражается с помощью некоторых материальных объектов (носителей), ассортимент и способ расположения которых задает информацию.Человек воспринимает сообщение посредством органов чувств. Приемник информации в технике воспринимает сообщения с помощью различной измерительной и регистрирующей аппаратуры. Носителем информации в различных информационных процессах может быть, например, камень, бумага, электрический кабель, магнитный диск.

Отображение множества состояний источника во множество состояний носителя называется способом кодирования. Таким образом, при выбранном способе кодирования какое-либо состояние заменяется своим образом — кодом состояния (или кодом информации, задаваемой этим состоянием). Так, мысли источника-человека могут быть закодированы определенным набором звуков, которые в свою очередь можно закодировать какими-то символами. Конечный набор знаков (символов) любой природы, из которых конструируются сообщения, образует алфавит некоторого языка.

Итак, последовательность символов алфавита, кодирующая состояние источника и воспринимаемая адресатом как сообщение, как информация, образует слово на этом языке. На передачу и переработку информации влияет то, сигналами какой природы отображается одна и та же информация, то есть каким кодом задана одна и та же информация. Если говорить о сигналах, дискретных по виду, то их множество конечно, поэтому их принято кодировать буквами алфавита того или иного естественного языка или цифрами той или иной системы счисления. Таким образом, дискретная информация отождествляется с алфавитно-цифровой, а простейшим алфавитом, достаточным для записи (представления) информации, является алфавит из двух символов, допустим 0 и 1.

**34. Требования к языкам программирования и их классификация**

**Машинно – ориентированные языки** – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Машинно –ориентированные языки позволяют использовать все возможности и особенности Машинно – зависимых языков:

- высокое качество создаваемых программ (компактность и скорость выполнения);

- возможность использования конкретных аппаратных ресурсов;

- предсказуемость объектного кода и заказов памяти;

- для составления эффективных программ необходимо знать систему команд и особенностифункционирования данной ЭВМ;

- трудоемкость процесса составления программ ( особенно на машинных языках и ЯСК), плохо защищенного отпоявления ошибок;

- низкая скорость программирования;

- невозможность непосредственного использования программ, составленных на этих языках, на ЭВМдругих типов.

Машинно-ориентированные языки по степени автоматического программирования подразделяются на классы.

*- Машинный язык*

Отдельный компьютер имеет свой определенный Машинный язык (далее МЯ), ему предписывают выполнение указываемых операций над определяемыми ими операндами, поэтому МЯ является командным.

*- Языки Символического Кодирования*

Языки Символического Кодирования (далее ЯСК),так же, как и МЯ, являются командными. Однако коды операций и адреса в машинных командах, представляющие собой последовательность двоичных (во внутреннем коде) или восьмеричных (часто используемых при написании программ) цифр, в ЯСК заменены на символы (идентификаторы), форма написания которых помогает программисту легче запоминать смысловое содержание операции. Это обеспечивает существенное уменьшение числа ошибок при составлении программ. Использование символических адресов – первый шаг к созданию ЯСК.

*- Автокоды*

Есть также языки, включающие в себя все возможности ЯСК, посредством расширенного введения макрокоманд - они называются Автокоды.

Макрокоманды обеспечивают передачу фактических параметров, которые в процессе трансляции вставляются в«остов» программы, превращая её в реальную машинную программу.

Развитые автокоды получили название Ассемблеры. Сервисные программы и пр., как правило, составлены на языках типа Ассемблер.

*- Макрос*

Язык, являющийся средством для замены последовательности символов описывающих выполнение требуемых действий ЭВМ наиболее сжатую форму - называется Макрос (средство замены).

В основном, Макрос предназначен для того, чтобы сократить запись исходной программы. Компонент программного обеспечения, обеспечивающий функционирование макросов, называется макропроцессором. На макропроцессор поступает макроопределяющий и исходный текст. Реакция макропроцессора на вызов-выдачу выходного текста.

Макрос одинаково может работать, как с программами, так и с данными.

**Машинно – независимые языки** – это средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ и ВС.

Подобные языки получили название высокоуровневых языков программирования. Программы, составляемые на таких языках, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка(задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на МЯ.

**Проблемно – ориентированные языки**

С расширением областей применения вычислительной техники возникла необходимость формализовать представление постановки и решение новых классов задач. Необходимо было создать такие языки программирования, которые, используя в данной области обозначения и терминологию, позволили бы описывать требуемые алгоритмы решения для поставленных задач, ими стали проблемно – ориентированные языки. Эти языки, языки ориентированные на решение определенных проблем, должны обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме.

Фортран, Алгол – языки, созданные для решения математических задач;

**Универсальные языки**

Универсальные языки были созданы для широкого круга задач: коммерческих, научных, моделирования и т.д.

**Диалоговые языки**

Появление новых технических возможностей поставило задачу перед системными программистами –создать программные средства, обеспечивающие оперативное взаимодействие человека с ЭВМ их назвали диалоговыми языками.

Задачи: управление и описание алгоритмов решения задач..

Одним из примеров диалоговых языков является Бэйсик.

Бэйсик использует обозначения подобные обычным математическим выражениям. Многие операторы являются упрощенными вариантами операторов языка Фортран. Поэтому этот язык позволяет решать достаточно широкий круг задач.

**Непроцедурные языки**

Непроцедурные языки составляют группу языков, описывающих организацию данных, обрабатываемых по фиксированным алгоритмам(табличные языки и генераторы отчетов), и языков связи с операционными системами.

Программы, составленные на табличном языке, удобно описывают сложные ситуации, возникающие при системном анализе.

**35. Понятие алгоритма, его основные свойства и способы записи**

Алгоритм – это точно определенная (однозначная) последовательность простых (элементарных) действий, обеспечивающих решение любой задачи из некоторого класса.

Свойства, которые характерны для алгоритмов :

-Дискретность алгоритма означает, что алгоритм разделен на отдельные шаги (действия), причем, выполнение очередного шага возможно только после завершения всех операций на предыдущем шаге. При этом набор промежуточных данных конечен и он получается по определенным правилам из данных предыдущего шага.

-Детерминированность алгоритма состоит в том, что совокупность промежуточных величин да любом шаге однозначно определяется системой величин, имевшихся на предыдущем шаге. Данное свойство означает, что результат выполнения алгоритма не зависит от того, кто (или что) его выполняет (т.е. от исполнителя алгоритма), а определяется только входными данными и шагами (последовательностью действий) самого алгоритма.

-Направленность алгоритма: если способ получения последующих величин из каких-либо исходных не приводит к результату, то должно быть указано, что следует считать результатом алгоритма.

-Массовость алгоритма: начальная система величин может выбираться из некоторого множества.

Исполнитель алгоритма – это субъект или устройство, способные правильно интерпретировать описание алгоритма и выполнить содержащийся в нем перечень действий.

Указания по выполнению действий для каждого исполнителя формулируются посредством некоторого языка, включающего набор служебных слов, обозначающих действия (команды), а также синтаксические правила их объединения.

Таким образом, при записи алгоритмов возможны ситуации, когда язык представления алгоритма является формальным, но в нем используются сложные команды, которые самим исполнителем переводятся на уровень истинно элементарных действий.

Для записи алгоритмов используют самые разнообразные средства. Выбор средства определяется типом исполняемого алгоритма. Выделяют следующие основные способы записи алгоритмов:   
- вербальный, когда алгоритм описывается на человеческом языке;   
- символьный, когда алгоритм описывается с помощью набора символов;   
- графический, когда алгоритм описывается с помощью набора графических изображений.

Общепринятыми способами записи являются графическая запись с помощью блок-схем и символьная запись с помощью какого-либо алгоритмического языка.

Описание алгоритма с помощью блок схем осуществляется рисованием последовательности геометрических фигур, каждая из которых подразумевает выполнение определенного действия алгоритма. Порядок выполнения действий указывается стрелками. Написание алгоритмов с помощью блок-схем регламентируется ГОСТом.

**36. Временная и обьемная сложность алгоритмов**

Под *временной сложностью алгоритма*(временем выполнения алгоритма) понимается время выполнения алгоритма, измеряемое в «шагах» (инструкциях алгоритма), которые необходимо выполнить для достижения запланированного результата на некоторой идеализированной вычислительной машине. Пусть  - алгоритм решения некоторого класса задач, и  определяет размерность входных данных (количество элементов массива, число вершин графа и т.п.). Определим  как время выполнения алгоритма в зависимости от входных данных. Временная сложность алгоритма характеризуется степенью роста . В большинстве случаев невозможно дать абсолютную оценку для числа шагов, требуемых для выполнения алгоритма. Точное число шагов может быть определено только для конкретной реализации.  обычно рассматривают как время выполнения алгоритма в наихудшем случае (как максимум времени выполнения по всем входным данным размера). Иногда рассматривают величину  как среднее в статистическом смысле время выполнения по всем входным данным размера . Следует отметить, что задача определения среднего времени выполнения гораздо сложнее нахождения времени выполнения в наихудшем случае, кроме того понятие «средних» данных имеет достаточно расплывчатое определение, поэтому на практике чаще всего используют величину . Говорят, что алгоритм *полиномиальный,* если  растет не быстрее, чем полином некоторой степени от  (при  алгоритмы называют *линейными*), в противном случае, выделяют *экспоненциальную*сложность. Если перед нами стоит задача определения максимального элемента линейного массива длины , то временная сложность такого алгоритма потребует  сравнений, здесь время выполнения не зависит от входных данных. Однако в большинстве случаев  существенно зависит от структуры начальных условий. Для оценки времени выполнения применяется асимптотическая символика. В 1894 г. П. Бахман предложил использовать символ  для описания степени роста функций. Говорят, что  имеет порядок , если. Константу пропорциональности  можно достаточно точно определить только для конкретной реализации алгоритма. Про алгоритмы, имеющие временную сложность, говорят, что они имеют порядок (степень) роста .

Другой важной характеристикой алгоритма является его емкость. Под емкостью принято понимать максимальный объем памяти, используемый при вычислении. Говоря о физическом пространстве, необходимом для алгоритма в общем случае, мы не можем определить его точно. Современная компьютерная техника предоставляет в распоряжение простого пользователя, такие объемы памяти, которые еще несколько лет назад казались недостижимыми. В этих условиях создается не совсем верное представление об определении затрат памяти, как излишней операции. Самый «быстрый» алгоритм решения задачи, требующий затрат памяти соизмеримых с предоставляемым объемом, в произвольном случае не может быть реализован на большинстве современных компьютеров. Определение емкости алгоритма тесно связано с условиями его реализации для конкретной ЭВМ и используемого языка программирования. С другой стороны, набор типов данных, используемых в реальных вычислениях, относительно постоянен. Среди скалярных величин это целочисленные типы, вещественные типы  и строковые типы. Среди векторных величин это, в основном, массивы скаляров, объем которых определяется совокупным объемом соответствующих скаляров. Большинство языков программирования выделяет для хранения скалярных величин один и тот же объем памяти.

**37. Последовательные, циклические и рекурсивные алгоритмы**

*Последовательные алгоритмы* – алгоритмы, в которых блоки выполняются последовательно друг за другом, в порядке заданной схемы.

*Циклические алгоритмы –* алгоритмы, в которых приходится многократно вычислять значения по одним и тем же математическим зависимостям (блок схемам) для различных значений входящих в них величин.

Функция (или процедура), которая прямо или косвенно обращается к себе, называется рекурсивной. При реализации рекурсивных алгоритмов каждый шаг рекурсии не дает непосредственного решения задачи, но сводит ее к такой же задаче меньшего размера. Этот процесс должен приводить к задаче такого размера, когда решение получается достаточно легко. Далее "обратный ход" дает последовательные решения для задачи все большего размера, вплоть до первоначального. В основе реализации процедуры с рекурсией лежит стек (память "магазинного" типа), где хранятся данные, участвующие во всех вызовах процедуры, при которых она еще не завершила свою работу.

**38. Задачи, требующие автоматизированной обработки информации**

Это задачи, в которых при обработки информации нельзя справится вручную, но одновременно нельзя обойтись без человека.

Например: Работа антивирусной программы;

Т.е. при проверке некоторых файлов и при нахождении в них вирусов обязательно вмешательство человека, т.к. именно он делает выбор дальнейшего поведения программы (удаление, карантин, лечение).

**39.Системы автоматизированной и автоматической обработки информации**

Системы автоматизированной обработки информации - это системы, в которых при обработки информации нельзя справится вручную, но одновременно нельзя обойтись без человека. К таким системам относятся все программы по работе человека в связке с ЭВМ (Программы по работам с базами данных).

Системы автоматической обработки информации - это системы, в которых при обработки информации не требуется непосредственное участие человека. Например: При проведении различного рода работ, возникает необходимость в устройстве, позволяющем в автоматическом режиме производить различного рода измерения, с последующей их передачей в центр обработки информации.

Следовательно, мы видим, что данные системы сильно связаны между собой, обычно начальную обработку проводят системы автоматической обработки информации и передают её системам, в которых уже необходимо принятие решения со стороны человека.

**40. Исторические этапы автоматизации обработки информации**

**Первый этап (1946, до конца1950-х).** Были созданы электронные цифровые компьютеры, которые строились на электронных лампах. Основной режим использования этих машин состоял в том, что математик, составивший программу, садился за пульт управления машиной и производил необходимые вычисления. В этот период началась интенсивная разработка средств автоматизации программирования, создание входных языков разных уровней, систем обслуживания программ, упрощающих работу на машине и увеличивающих эффективность её использования.

**Второй этап (до середины 1960-х).** С технической точки зрения период чётко очерчен переходом на полупроводники, с точки зрения ЭВМ второго поколения характеризуются расширенными возможностями по вводу-выводу, увеличенным объёмом запоминающих устройств, развитыми системами программирования. Первые ОС просто автоматизировали работу оператора ЭВМ, связанную с выполнением задания пользователя: ввод в ЭВМ текста программы, вызов нужного транслятора, вызов необходимых библиотечных программ и т.д. Теперь же вместе с программой и данными в ЭВМ вводится ещё и инструкция, где перечисляются этапы обработки и приводится ряд сведений о программе и её авторе. Затем в ЭВМ стали вводить сразу по нескольку заданий пользователей (пакет заданий), ОС стали распределять ресурсы ЭВМ между этими заданиями – появился мультипрограммный режим обработки.

**Третий этап (до начала 1970-х).** Базой ЭВМ третьего поколения являются интегральные схемы, состоящей из десятков электронных элементов, образованных в прямоугольной пластине кремня с длиной стороны не более 1 см, позволило увеличить быстродействие и надёжность ЭВМ на их основе, а также уменьшить габариты, потребляемую мощность и стоимость ЭВМ. Машины третьего поколения имеют развитые операционные системы, обладают возможностями мультипрограммирования, т.е. одновременного выполнения нескольких программ. Многие задачи управления памятью, устройствами и ресурсами стала брать на себя операционная система или же непосредственно сама машина. В период машин третьего поколения произошёл крупный сдвиг в области применения ЭВМ. Если раньше ЭВМ использовалась в основном для научно-технических расчётов, то в 1960-1970-е гг. всё больше места стала занимать обработка символьной информации.

**Четвёртый этап (по настоящее время).** На первый план вышла задача экономии человеческих, а не машинных ресурсов. Существовавшая концепция первых этапов информационной технологии постепенно была заменена на новую: «всё, что могут делать машины должны делать машины; люди выполняют лишь ту часть работы, которую нельзя автоматизировать». С точки зрения структуры машины этого поколения представляют собой многопроцессорные и многомашинные комплексы, работающие на общую память и общее поле внешних устройств. Для этого периода характерно широкое применение систем управления базами данных, компьютерных сетей, систем распределённой обработки данных.

Последующие поколения ЭВМ будут представлять, по-видимому, оптоэлектронные ЭВМ с массовым параллелизмом и нейронной структурой – с распределённой сетью большого числа (десятки тысяч) несложных процессоров, моделирующих структуру нейронных биологических систем, произойдёт качественный переход от обработки данных к обработке знаний.

С каждым новым поколением ЭВМ увеличивались быстродействия и надежность их работы при уменьшении стоимости и размеров, совершенствовались устройства ввода и вывода информации. В соответствии с трактовкой компьютера - как технической модели информационной функции человека - устройства ввода информации (зрительному, звуковому) и, следовательно, операция по ее вводу в компьютер становится все более удобной для человека.

**41. Базовая структура цифровой машины с хранимой программой**

Итак, в структурном плане ЭВМ представляет собой систему, состоящую из пяти основных функциональных блоков: устройства ввода информации, памяти, устройства управления, АЛУ и устройства вывода информации. Информация, вводимая в центральный процессор, представляет собой или программные команды или данные, которые нужно обработать. Возможность хранения программ в памяти и относительно большие размеры отличают ЭВМ от микрокалькуляторов.

Микропроцессор координирует работу всех устройств рассматриваемой цифровой системы с помощью шины управления. Помимо шины управления имеется адресная шина (16 параллельных проводников), которая служит для выбора определенной ячейки памяти, порта ввода или порта вывода. По информационной шине, или шине данных (8 параллельных проводников) осуществляется двунаправленная пересылка данных к микропроцессору и от микропроцессора. Важно отметить, что микропроцессор может посылать информацию в память ЭВМ или к одному из портов вывода, а также получать информацию из памяти или от одного из портов ввода.

ПЗУ в ЭВМ обычно содержит некоторую программу. Программа - это список особым образом закодированных команд, задающих точную последовательность выполняемых микропроцессором операций. На практике ПЗУ содержит запускающую программу (программу инициализации) и, возможно, некоторые другие программы. Программы могут быть также загружены в ЗУПВ из внешнего ЗУ. Это программы пользователя.

ЗУПВ в схеме показано как память для хранения данных. Здесь находятся данные, используемые в процессе выполнения конкретной программы.

**42. Архитектура и структура команд четырёхадресной машины**

Команда к процессору ЭВМ первоначально содержала следующую информацию:

1. Код операции, указывающий на операцию, которую должна выполнить ЭВМ (сложение, вычитание, и т.п.)
2. Адрес первой ячейки памяти (первый операнд),
3. Адрес второй ячейки памяти (второй операнд)
4. Адрес ячейки памяти, куда нужно поместить результат
5. Адрес следующей команды алгоритма.

Адресная часть описывает, где используемая информация хранится и куда поместить результат.

Итого пять полей, из них один код операции (КОП) и четыре адреса.

При такой структуре ЭВМ первый операнд приходится сохранять в процессоре, так как нельзя одновременно прочесть из памяти оба операнда. Ячейка, в которую сохраняется значение, называется аккумулятором.

В четырехадресной команде на каждый адрес приходится по 2 бита, то есть можно адресовать только 23=8 ячеек памяти. В современных ЭВМ используется намного большее количество ячеек, поэтому четырехадресные команды не применяются.

**43. Архитектура и структура команд трёхадресной машины**

Если команды размещать друг за другом в памяти, то можно не указывать в команде адрес следующей. Таким образом получаются трехадресные команды. При их использовании также вводятся специальные команды перехода, которые используются там, где нарушается естественная последовательность команд. Например при условии if, по которому нужно перейти на другую команду, пропустив несколько. Команда перехода – одноадресная: она состоит всего из двух полей: кода операции и адреса команды, на которую осуществляется переход.

**44. Архитектура и структура команд двухадресной машины**

Трехадресная команда легко расшифровывалась и была удобна в использовании, но с ростом объемов ОЗУ ее длина становилась непомерно большой. Для технической реализации использование трехадресных машин стало неудобным, поэтому появились двухадресные машины, длина команды в которых сокращалась за счет исключения адреса записи результата. В таких ЭВМ результат операции оставался в специальном регистре (**сумматоре**) и был пригоден для использования в последующих вычислениях. В некоторых машинах результат записывался вместо одного из операндов.

**45. Архитектура и структура команд одноадресной машины**

Часто результат выполнения предыдущей команды используется как операнд для следующей. В таких случаях можно использовать одноадресные команды – они будут содержать в себе только КОП и адрес второго операнда. Если пользоваться только одноадресными командами, их можно поделить на группы:

арифметической или логической обработки. Первое значение берется с указанного адреса, второе из аккумулятора. С ними совершается действие соответствующее коду команды, а затем результат записывается в аккумулятор.

пересылки. Дают приказ на обмен информации между аккумулятором и памятью (запись в указанную ячейку данных из аккумулятора или наоборот, запись в аккумулятор данных из ячейки памяти)

передачи управления. Обеспечивает переход (возможно по некоторому условию) к команде из указанной ячейки памяти (такая команда уже описана в трехадресной машине)

ввод-вывод

**46. Реализация устройств оперативной и долговременной памяти**

**Память** - устройство для хранения информации в виде данных и программ. Память делится прежде всего на внутреннюю (расположенную на системной плате) и внешнюю (размещенную на разнообразных внешних носителях информации).

**Внутренняя память** в свою очередь подразделяется на:

**- ПЗУ** (постоянное запоминающее устройство) или ROM (read only memory), которое содержит - постоянную информацию, сохраняемую даже при отключенном питании, которая служит для тестирования памяти и оборудования компьютера, начальной загрузки ПК при включении. Запись на специальную кассету ПЗУ происходит на заводе фирмы-изготовителя ПК и несет черты его индивидуальности. **Объем** ПЗУ относительно невелик - от 64 Кб до нескольких Мб.

**- ОЗУ** (оперативное запоминающее устройство, ОП — оперативная память) или RAM (random access memory), служит для оперативного хранения программ и данных, сохраняемых только на период работы ПК. Она энергозависима, при отключении питания информация теряется.

**Объем** ОП колеблется в пределах от 64 Кб до 1024 Мб и выше, как правило, ОП имеет модульную структуру и может расширяться за счет добавления новых микросхем.

**Кэш-память** - имеет малое время доступа, служит для временного хранения промежуточных результатов и содержимого наиболее часто используемых ячеек ОП и регистров МП.

**Объем** кэш-памяти зависит от модели ПК и составляет обычно 512 до 1024 Кб.

**47. Классификация устройств памяти систем обработки информации**

В состав памяти ЭВМ входят также ЗУ, **принадлежащие отдельным функциональным блокам** компьютера. Формально эти устройства непосредственно не обслуживают основные потоки данных и команд, проходящие через процессор. Их назначение обычно сводится к буферизации данных, извлекаемых из каких-либо устройств и поступающих в них.

Типичным примером такой памяти является **видеопамять** графического адаптера, которая используется в качестве буферной памяти для снижения нагрузки на основную память и системную шину процессора.

Другими примерами таких устройств могут служить **буферная память** контроллеров жестких дисков, а также память, использовавшаяся в каналах (процессорах) ввода-вывода для организации одновременной работы нескольких внешних устройств.

Емкости и быстродействие этих видов памяти зависят от конкретного функционального назначения обслуживаемых ими устройств. Для видеопамяти, например, объем может достигать величин, сравнимых с оперативными ЗУ, а быстродействие – даже превосходить быстродействие последних.

Следующей ступенью памяти, ставшей фактически стандартом для любых ЭВМ, являются **жесткие диски**. В этих ЗУ хранится практически вся информация, которая используется более или менее активно, начиная от операционной системы и основных прикладных программ и кончая редко используемыми пакетами и справочными данными.

Емкость этой ступени памяти, которая может включать в свой состав до десятков дисков, обеспечивая хранение очень большого количества данных, зависит от области применения ЭВМ. Типовая емкость жесткого диска, составляющая на начало 2000-х годов десятки гигабайт, удваивается примерно каждые полтора года.

**Внешняя память.** Устройства внешней памяти весьма разнообразны. Предлагаемая классификация учитывает тип **носителя**, т.е. материального объекта, способного хранить информацию.

**Диски** относятся к носителям информации с прямым доступом, т.е. ПК может обратиться к дорожке, на которой начинается участок с искомой информацией или куда нужно записать новую информацию, непосредственно.

**Магнитные диски (МД)—** в качестве запоминающей среды используются магнитные материалы со специальными свойствами, позволяющими фиксировать два направления намагниченности. Каждому из этих состояний ставятся в соответствие двоичные цифры — 0 и 1. При записи и чтении МД вращается вокруг своей оси, а механизм управления магнитной головкой подводит ее к выбранной для записи или чтения дорожке.

Данные на дисках хранятся в **файлах** — именованных областях внешней памяти, выделенных для хранения массива данных.

«Винчестеры» изготовлены из сплавов алюминия или из керамики и покрыты ферролаком, вместе с блоком магнитных головок помещены в герметически закрытый корпус. **Емкость** накопителей за счет чрезвычайно плотной записи достигает нескольких гигабайт, быстродействие также выше, чем у съемных дисков (за счет увеличения скорости вращения, т.к. диск жестко закреплен на оси вращения).

**Дисковые массивы RAID** - применяются в машинах-серверах БД и в суперЭВМ, они представляют собой матрицу с резервируемыми независимыми дисками, несколько НЖМД объединены в один логический диск. **Емкость** таких дисков составляет до 5Т6 (терабайт=1012).

**НОД** (накопители на оптических дисках) делятся на:

**- не перезаписываемые** лазерно-оптические диски или компакт-диски (CD-ROM). Поставляются фирмой-изготовителем с уже записанной на них информацией. Запись на них возможна в лабораторных условиях лазерным лучом большой мощности. В оптическом дисководе ПК эта дорожка читается лазерным лучом меньшей мощности. Ввиду чрезвычайно плотной записи CD-ROM имеют емкость до 1,5 Гб.

**- перезаписываемые** CD-диски имеют возможность записывать информацию прямо с ПК, но для этого необходимо специальное устройство.

**49. Представление целых и действительных чисел в позиционных системах счисления**

Каждая позиционная система счисления определяется некоторым [**числом**](http://ru.wikipedia.org/wiki/Число) *b* > 1 (т. н. **основание системы счисления**) таким, что *b* единиц в каждом разряде объединяется в одну единицу следующего по старшинству разряда. Система счисления с основанием *b* также называется ***b*-ричной**.

[Число](http://ru.wikipedia.org/wiki/Целое_число) *x* в *b*-ричной системе счисления представляется в виде конечной [линейной комбинации](http://ru.wikipedia.org/wiki/Линейное_пространство) степеней числа *b*:

, где *ak* — это целые числа, называемые **цифрами**, удовлетворяющие неравенству .

Каждая степень *bk* в такой записи называется *b*-ричным **разрядом**, старшинство разрядов и соответствующих им цифр определяется значением показателя *k*. Обычно для ненулевого числа *x* требуют, чтобы старшая цифра *an*− 1 в *b*-ричном представлении *x* была также ненулевой.

Если не возникает разночтений (например, когда все цифры представляются в виде уникальных письменных знаков), число *x* записывают в виде последовательности его *b*-ричных цифр, перечисляемых по убыванию старшинства разрядов слева направо:



**48. Позиционные системы счисления и выполнение арифметических операций**

*Системой счисления* называется совокупность приемов наименования и записи чисел. В любой системе счисления для представления чисел выбираются некоторые символы (их называют *цифрами*), а остальные числа получаются в результате каких-либо операций над цифрами данной системы счисления.

Система называется *позиционной*, если значение каждой цифры (ее вес) изменяется в зависимости от ее положения (позиции) в последовательности цифр, изображающих число.

Число единиц какого-либо разряда, объединяемых в единицу более старшего разряда, называют *основанием позиционной системы счисления*. Если количество таких цифр равно *P*, то система счисления называется *P*-ичной. Основание системы счисления совпадает с количеством цифр, используемых для записи чисел в этой системе счисления.

Запись произвольного числа *x* в *P*-ичной позиционной системе счисления основывается на представлении этого числа в виде многочлена

*x = anPn + an*-1*Pn*-1*+ ... + a*1*P*1*+ a*0*P*0*+ a*-1*P*-1*+ ... + a-mP-m*

Арифметические действия над числами в любой позиционной системе счисления производятся по тем же правилам, что и десятичной системе, так как все они основываются на правилах выполнения действий над соответствующими многочленами. При этом нужно только пользоваться теми таблицами сложения и умножения, которые соответствуют данному основанию *P* системы счисления.

При переводе чисел из десятичной системы счисления в систему с основанием *P* > 1 обычно используют следующий алгоритм:

1) если переводится целая часть числа, то она делится на *P*, после чего запоминается остаток от деления. Полученное частное вновь делится на *P*, остаток запоминается. Процедура продолжается до тех пор, пока частное не станет равным нулю. Остатки от деления на *P* выписываются в порядке, обратном их получению;

2) если переводится дробная часть числа, то она умножается на *P*, после чего целая часть запоминается и отбрасывается. Вновь полученная дробная часть умножается на *P* и т.д. Процедура продолжается до тех пор, пока дробная часть не станет равной нулю. Целые части выписываются после двоичной запятой в порядке их получения. Результатом может быть либо конечная, либо периодическая двоичная дробь. Поэтому, когда дробь является периодической, приходится обрывать умножение на каком-либо шаге и довольствоваться приближенной записью исходного числа в системе с основанием *P*.

**50. Связь между системами счисления с основаниями вида 2K**

На ранних этапах развития вычислительной техники программы писали в машинных кодах, то есть без использования языков программирования. Для обозначение кодов операций машина оперирует с довольно длинными двоичными числами. Программисту трудно было работать с таким количеством знаков. Поэтому стали использовать системы счисления, которые с одной стороны относительно малозначны. А с другой обеспечивают легкий перевод чисел в двоичную систему и обратно. Такими системами являются системы, родственные двоичной.

Система называется родственной двоичной, если ее основание является степенью числа 2. К таким системам относятся четверичная, восьмеричная и шестнадцатеричная.

Если посмотреть на двоичное число, представляющее собой представление некоторого десятичного с весьма умеренным числом цифр (например, трёх или четырёхзначного числа), то обнаружится, что выглядит это чрезмерно длинно. Более того, длинная “однообразная” цепочка из нулей и единиц очень плохо воспринимается глазами. Чтобы облегчить ситуацию, для более компактной записи используется восьмеричная или шестнадцатеричная система счисления. Особенностью данных оснований является тот факт. Что и 8, и 16 есть степени двойки, а значит, перевод между ними и двоичной системой максимально прост. Учитывая, что 8 = 23, а 16 = 24, получаем, что каждая восьмеричная цифра объединяет ровно 3 двоичных разряда, а шестнадцатеричная – 4.

Отсюда немедленно следует алгоритм перевода из двоичной системы в восьмеричную (шестнадцатеричную):

* 1. сгруппировать двоичные разряды **справа налево**по три (четыре); если в старшей (т.е. самой левой) группе битов не хватает, их можно дополнить слева незначащими нулями;
* 2. заменить каждую из полученных групп соответствующей ей восьмеричной (шестнадцатеричной) цифрой.

**51. Перевод целых и действительных чисел из одной позиционной системы в другую**

Поскольку с математической точки зрения системы счисления с любыми основаниями равноправны, существует единый алгоритм перевода чисел из одной системы счисления в другую. Он заключается в последовательном делении рассматриваемого числа на основание системы счисления. К сожалению, алгоритм требует проведения арифметических действий в той системе счисления, в которой представлено исходное число, поэтому удобен лишь для перевода из десятичной системы в произвольную, но не наоборот.

Частным случаем указанного выше способа является перевод из десятичной системы счисления в двоичную, который нужен, чтобы узнать представление в компьютере произвольного десятичного числа.

Что касается обратного перевода из двоичной системы в десятичную, то универсальный алгоритм деления на основание системы здесь также возможен. Но. Как уже говорилось, его непосредственная арифметическая реализация неудобна. Поэтому на практике используется иной алгоритм, базирующийся на другом универсальном свойстве, о котором уже упоминалось в связи с определением основания системы счисления. Речь идёт о том, что запись произвольного числа в любой системе счисления суть его разложения по степеням основания.

**52. Прямой, обратный и дополнительный коды**

Целые числа со знаком обычно занимают в памяти компьютера один, два или четыре байта.  Для хранения целых чисел со знаком старший (левый) разряд в машинном слове отводится под знак числа (если число положительное, то в знаковый разряд записывается ноль, если число отрицательное – единица). Ровно половина из всех 2nчисел будут отрицательными; учитывая необходимость нулевого значения, положительных будет на единицу меньше.

Максимальное положительное число (с учетом выделения одного разряда на знак) для целых чисел со знаком в n-разрядном представлении равно . Минимальное отрицательное  число (с учетом выделения одного разряда на знак) для целых чисел со знаком в n-разрядном представлении равно -.

Диапазоны значений целых чисел со знаком:

-         в 8-разрядной ячейке: от -128 до 127;

-         в 16-разрядной ячейке: от -32 768 до 32 767;

-         в 32-разрядной ячейке: от -2 147 483 648 до 2 147 483 647.

   Для представления отрицательного числа используется *дополнительный код.* Дополнительный код положительного числа совпадает с его прямым кодом.

*Прямой код*целого положительного числа может быть получен следующим образом: число переводится в двоичную систему счисления, а затем его двоичную запись слева дополняют необходимым количеством нулей в соответствии с разрядностью машинного слова.

Для записи внутреннего представления целого число со знаком (-А) необходимо:

1)   модуль числа записать в ***прямом коде***в n двоичных разрядах;

2)   получить ***обратный код*** числа, для этого значения всех  бит инвертировать – все единицы заменить на нули  и все нули заменить на единицы);

3)  к полученному обратному коду прибавить единицу. Получим *дополнительный код* целого числа со знаком.

**53. Нормализованное представление чисел, переполнение, модифицированные коды**

Представление числа с плавающей запятой

В общем случае имеет вид:   
A = m \* N p,  
где N - основание системы счисления,   
p - целое число, называемое порядком числа A,   
m - мантисса числа A (¦m¦<1).   
Так как в ЭВМ применяется двоичная система счисления, то естественно было бы:   
A = m \* 2p,

Однако, при таком способе мы получили бы слишком маленький диапазон чисел, либо пришлось бы использовать много разрядов для хранения порядка. Поэтому в современных процессорах используется не число 2, а число 16:

A = m \* 16 p,

Код числа отображается только в виде порядка и мантиссы, представленных в двоичной форме.   
Двоичное число называется нормализованным, если его мантисса удовлетворяет неравенству   
1/16< ¦ m ¦ < 1 .   
Неравенство показывает, что двоичное число является нормализованным, если в старшем разряде мантиссы стоит единица.  
Диапазон представления нормализованных двоичных чисел, взятых по абсолютному значению, удовлетворяет неравенству:   
16-1\*16-(16k-1) < A < (1 - 16-l)\*1616k-1,   
где l - число разрядов мантиссы;   
k - число разрядов порядка;   
16-1 - наименьшее значение нормализованной мантиссы;   
1 - 16-l - наибольшее значение нормализованной мантиссы.   
Широкий диапазон представления чисел с плавающей запятой удобен для научных и инженерных расчетов. Для повышения точности вычислений во компьютерах предусмотрена возможность использования формата двойной длины, однако при этом происходит увеличение затрат памяти на хранение данных и замедляются вычисления.

При сложении чисел, меньших единицы, в машине могут быть получены числа, по абсолютной величине большие единицы. Для обнаружения *переполнения* разрядной сетки в ЭВМ применяются *модифицированные* прямой, обратный и дополнительный коды. В этих кодах знак кодируется двумя разрядами, причем знаку "плюс" соответствует комбинация 00, а знаку "минус" - комбинация 11.   
Правила сложения для модифицированных кодов те же, что и для обычных. Единица переноса из старшего знакового разряда в модифицированном дополнительном коде отбрасывается, а в модифицированном обратном коде передается в младший цифровой разряд.   
Признаком *переполнения* служит появление в знаковом разряде суммы комбинации 01 при сложении положительных чисел (положительное переполнение) или 10 при сложении отрицательных чисел (отрицательное переполнение). Старший знаковый разряд в этих случаях содержит истинное значение знака суммы, а младший является старшей значащей цифрой числа. Для коррекции переполнения число нужно сдвинуть в разрядной сетке на один разряд вправо, а в освободившийся старший знаковый разряд поместить цифру, равную новому значению младшего знакового разряда. После корректировки переполнения мантиссы результата необходимо увеличить на единицу порядок результата.

**54. Алгоритмы сложения и вычитания в дополнительных и обратных кодах**

В ЭВМ прямой код применяется только для представления положительных двоичных чисел. Для представления отрицательных чисел применяется либо дополнительный, либо обратный код, так как над отрицательными числами в прямом коде неудобно выполнять арифметические операции.

Формула для образования дополнительного кода дроби:   
[A]доп = 10 + A.

Формула для образования обратного кода дроби:   
[A]обр = 10 - 10-(n-1) + A.

Формула для образования дополнительного кода целого числа:   
[A]доп = 10n + A.

Формула для образования обратного кода целого числа:   
[A]обр = 10n - 1 + A.

Таким образом, правила для образования дополнительного и обратного кода состоят в следующем:   
- для образования дополнительного кода отрицательного числа необходимо в знаковом разряде поставить единицу, а все цифровые разряды инвертировать (заменить 1 на 0, а 0 - на 1), после чего прибавить 1 к младшему разряду;   
- для образования обратного кода отрицательного числа необходимо в знаковом разряде поставить единицу, а все цифровые разряды инвертировать.

Примечание: при данных преобразованиях нужно учитывать размер разрядной сетки.

Прямой код можно получить из дополнительного и обратного по тем же правилам, которые служат для нахождения дополнительного и обратного кодов.

Замена вычитания двоичных чисел A1 - A2 сложением с дополнениями [A1]пр + [-A2]доп или [A1]пр + [-A2]обр позволяет оперировать со знаковыми разрядами так же, как и с цифровыми. При этом перенос из старшего знакового разряда, если он возникает, учитывается по разному для обратного и дополнительного кодов:   
- при использовании дополнительного кода единица переноса из знакового разряда отбрасывается;   
- при использовании обратного кода единица переноса из знакового разряда прибавляется к младшему разряду суммы (осуществляется так называемый циклический перенос).

Если знаковый разряд результата равен нулю, то в результате получено положительное число, которое представлено в прямом коде. Если в знаковом разряде единица, то результат отрицательный и представлен в обратном или дополнительном коде.

**55. Алгоритмы умножения в дополнительных и обратных кодах**

Умножение в двоичной системе - это сложение и сдвиг.

*Общий алгоритм умножения целых двоичных знаковых чисел, представленных в дополнительном (обратном) коде:*

1) Исходное значение суммы частичных произведений (СЧП) принимаем равным 0 и значение счетчика тактов (СчТ) учтанавливаем равным *n* - колличеству разрядов множимого.

2) Анализируется младший разряд цифры множителя. Если он равен 1, то к СЧП прибавляем множимое в том коде, в котором оно представлено (сложение производится совмещенно по старшим разрядам). Если анализируемый разряд равен 0, то сложение не производится.

3) Осуществляется арифметический сдвиг вправо СЧП с учетом флагов переноса CF и OF. СчТ уменьшается на 1.

4) Пункты 2 и 3 последовательно выполняются для всех цифровых разрядов множителя, начиная с младшего.

5) Если множитель - положительное число, полученный результат - произведение. Если множитель - отрицательное число, то для получения произведения к результату прибавляется множимое с обратным знаком. Сложение совмещается по старшему разряду. Полученное произведение будет в дополнительном (обратном) коде.

**56. Последовательность обработки прикладных программ**

**Программирование** (programming) - теоретическая и практическая деятельность, связанная с созданием программ. Решение задач на компьютере включает в себя следующие основные этапы, часть из которых осуществляется без участия компьютера.

**1. Постановка задачи:**

• сбор информации о задаче;

• формулировка условия задачи;

• определение конечных целей решения задачи;

• определение формы выдачи результатов;

• описание данных (их типов, диапазонов величин, структуры и т. п.).

**2. Анализ и исследование задачи, модели:**

• анализ существующих аналогов;

• анализ технических и программных средств;

• разработка математической модели;

• разработка структур данных.

**3. Разработка алгоритма:**

• выбор метода проектирования алгоритма;

• выбор формы записи алгоритма (блок-схемы, псевдокод и др.);

• выбор тестов и метода тестирования;

• проектирование алгоритма.

**4. Программирование:**

• выбор языка программирования;

• уточнение способов организации данных;

• запись алгоритма на выбранном языке

программирования.

**5. Тестирование и отладка:**

• синтаксическая отладка;

• отладка семантики и логической структуры;

• тестовые расчеты и анализ результатов тестирования;

• совершенствование программы.

**6. Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2-5.**

**7. Сопровождение программы:**

• доработка программы для решения конкретных задач;

• составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

**57. Жизненный цикл программного обеспечения**

**Системное программное обеспечение:** Комплекс аппаратно - программных средств расширяющих функциональные возможности операционной системы.

Все программы можно разбить на два класса по характеру использования:

- утилитарные программы — предназначены для удовлетворения нужд их разработчиков, программы «для себя»;

- программные продукты — предназначены для удовлетворения потребностей пользователей, широкого распространения и продажи.

Программный продукт должен быть соответствующим образом подготовлен к эксплуатации, иметь необходимую техническую документацию, предоставлять сервис и гарантию надежной работы программы, иметь товарный знак изготовителя. Только при таких условиях созданный программный комплекс может быть назван **программным продуктом.**

Программный продукт имеет несколько качественных характеристик:

-         алгоритмическая сложность;

-         полнота функций обработки;

-         объём файлов программ;

-         требования к операционной системе и техническим средствам обработки со стороны программного средства;

-         объём дисковой памяти;

-         размер оперативной памяти.

Показатели качества должны содержать следующие аспекты:

-         насколько хорошо можно использовать программный продукт (просто, надёжно, эффективно);

-         насколько легко эксплуатировать программный продукт;

-         можно ли использовать программный продукт при изменении условия его применения.

В условиях существования рынка программных продуктов важными характеристиками являются стоимость, количество продаж, время нахождения на рынке, известность фирмы-производителя и самой программы, наличие на рынке программных продуктов аналогичного назначения.

Программный продукт любого вида характеризуется жизненным циклом, состоящим из отдельных этапов.

**Маркетинг** предназначен для изучения требований к создаваемому программному продукту (технических, программных, пользовательских). Изучаются также существующие аналоги и продукты-конкуренты. Оцениваются необходимые для разработки материальные, трудовые и финансовые ресурсы, а также устанавливаются примерные сроки разработки.

**Проектирование структуры** — алгоритмизация процесса обработки данных, детализация функций, разработка архитектурного проекта, выбор методов и средств создания программ.

**Программирование, тестирование и отладка** — основной этап работы по разработке программного средства. Часто отдельные работы этого этапа ведутся параллельно, что позволяет сократить общее время разработки.

**Документирование** — обязательный вид работы. Документация должна содержать необходимые сведения по установке, обеспечению надёжной работы продукта, справочное пособие для пользователя, демонстрационные версии, примеры документов, создаваемых при помощи данного программного продукта, обучающие программы.

**Выход программного продукта на рынок** связан с организацией продаж массовому пользователю. Здесь применяются стандартные методы — реклама, увеличение числа каналов реализации, создание дилерской и дистрибьюторской сети, гибкая ценовая политика.

**Эксплуатация и сопровождение** идут, как правило, параллельно. В процессе эксплуатации могут выявляться ошибки, и устранение этих ошибок ведётся в режиме сопровождения, то есть оказание сервисной помощи, обеспечение новыми версиями программ, организация «горячих телефонных линий» для консультаций.

**Снятие программного продукта с продажи** и отказ от его сопровождения происходит, как правило, в случае изменения технической политики фирмы-изготовителя, неэффективности работы программного продукта, наличия в нём неустранимых ошибок, отсутствие спроса.

Длительность жизненного цикла разных программных продуктов неодинакова. Для большинства современных программ его длительность составляет 2-3 года. Хотя часто встречаются на компьютерах и давно снятые с производства программные продукты.

**58. Задачи системного программного обеспечения**

Системное ПО включает программы, необходимые для согласования работы всего вычислительного комплекса при решении различных задач, а также при разработке новых программ.

По функциональному назначению в системном ПО выделяют операционную систему (ОС), систему программирования (СП), системные обслуживающие программы (утилиты), средства контроля и диагностики устройств компьютера.

Системные обслуживающие программы  
Системные обслуживающие программы предназначены для выполнения типовых действий по подготовке носителей информации (магнитных дисков и лент) к записи на них данных, копирования, переименования и удаления файлов, восстановления поврежденной информации и т. п. В ряде случаев системные обслуживающие программы считают частью ОС.

Важным классом системных программ являются драйверы. Они расширяют возможности ОС, например, позволяя ей работать с тем или иным внешним устройством, обучая ее новому протоколу обмена данными и т. д. Так, первоначально попавшие в нашу страну версии DOS, Windows и OS/2 были английскими и не поддерживали ввод русских буквы с клавиатуры. Поэтому различные программисты создали драйверы, обеспечивающие эти средства.

Большинство ОС содержит немало драйверов в комплекте своей поставки, и программа установки ОС устанавливает (задействует) те драйверы, которые нужны для поддержки устройств и функций ОС, указанных пользователем. Драйверы для различных ОС часто поставляются и вместе с новыми устройствами или контроллерами.

Весьма популярный класс системных программ составляют программы-оболочки. Они обеспечивают более удобный и наглядный способ общения с компьютером, чем штатные средства ОС. Многие пользователи настолько привыкли к удобствам, предоставляемым своей любимой программой-оболочкой, что чувствуют себя без нее «не в своей тарелке».

К системным программам можно также отнести большое количество так называемых утилит, т. е. программ вспомогательного назначения. Чаще всего используются следующие типы утилит:  
• программы резервирования — позволяют быстро скопировать нужную для Вас информацию, находящуюся на жестком диске компьютера, на дискеты, съемные диски или кассеты стримера;  
• антивирусные программы — предназначены для предотвращения заражения компьютерным вирусом и ликвидации последствий заражения;  
• программы-упаковщики (архиваторы) позволяют за счет применения специальных методов «упаковки» информации сжимать информацию на дисках, т. е. создавать копии файлов меньшего размера, а также объединять копии нескольких файлов в один архивный файл.

**59. Назначение и структура операционных систем**

Операционная система является программной надстройкой над архитектурой компьютера, которая обеспечивает удобный пользовательский интерфейс, берет на себя функции автоматического управления рядом его подсистем и предоставляет готовые процедуры управления внутренними и внешними ресурсами. То есть, операционная система является некоей автоматизированной системой управления работой и ресурсами компьютера, повышающая удобство и эффективность его использования.

**Операционная система** (ОС) - неотъемлемая часть большинства современных вычислительных систем, управляющая всеми его аппаратными компонентами. Другими словами, Операционная (супервизорная, мониторная, исполнительная) Система (ОС) - есть организованная совокупность программ (систем), которая действует как интерфейс между аппаратурой ЭВМ и пользователями. Она обеспечивает пользователей набором средств для облегчения проектирования, программирования, отладки и сопровождения программ и в то же время управляет распределением ресурсов для обеспечения эффективной работы.

Структуру ОС составляют следующие модули:

*базовый модуль* (ядро ОС) - управляет работой программ и файловой системой, обеспечивает доступ к ней и обмен файлами между периферийными устройствами;

*командный процессор* - расшифровывает и исполняет команды пользователя, поступающие прежде всего через клавиатуру;

*драйверы периферийных устройств* - программно обеспечивают согласованность работы этих устройств с процессором (каждое периферийное устройство обрабатывает информацию по-разному и в различном темпе);

*дополнительные сервисные программы* (утилиты) - делают удобным и многосторонним процесс общения пользователя с компьютером.

В общем случае операционная система выполняет следующие четыре функции:

предоставляет другим программам определенный вид обслуживания (посредством программ-утилит), например выделение и назначение памяти, синхронизацию процесса вычислений и организацию взаимосвязи между различными процессами в вычислительной системе;

обеспечивает защиту (в определенной мере) других программ от последствий различных особых ситуаций, возникающих при машинной реализации данной программы, таких, как прерывания и машинные сбои;

реализует с той или иной степенью сложности принцип “виртуальной машины”, что позволяет группе программ использовать общие вычислительные ресурсы, например процессор (процессоры) и основную память;

организует и следит за выполнением принципов управления при решении таких задач, как обеспечение защиты данных от несанкционированного доступа и реализация системы приоритетов доступа программ к вычислительным ресурсам.

**Графический интерфейс пользователя**

Графический интерфейс пользователя изначально был несвойственен неигровым программам, однако, будучи призванным облегчить общение пользователя с компьютером и программой, хорошо прижился на IBM PC и стал неотъемлемой частью любой уважающей себя операционной системы.

Оболочка Microsoft Windows не была изначально операционной системой, она возникла в виде стандартизатора графического интерфейса и прижилась исключительно потому, что пользователь хотел видеть программу, с которой ему часто приходится работать красивой, практичной, удобной и легкой в освоении и использовании.

**60. Назначение и последовательность функционирования компиляторов**

Компьютерные программы, или *программное обеспечение,* представляют собой файлы, содержащие инструкции, которые указывают компьютеру, что следует делать.

*Программирование* представляет собой процесс определения последовательности инструкций, которые должен выполнить компьютер для решения определенной задачи. Для указания этих инструкций мы используем *язык программирования* C++.

Система должна выделять место в памяти для написания текста. С помощью текстового редактора мы вносим программные операторы в *исходный файл.* Потом мы запускаем программу, различающую текст, т.е. используем — *компилятор —* для преобразования операторов из формата, который мы можем читать и понимать, в единицы и нули, которые понимает компьютер. Переводом текста в машинные команды занимаются программы-трансляторы. Функциями трансляторов являются либо ликвидация неполадок, либо предупреждение о совершенных ошибках. Основной задачей является оптимизация текста.

Современные трансляторы являются компиляторами, следовательно, мы можем сформулировать последовательность функционирования компилятора:

-сначала компилятор проверяет программу на наличие ошибок и сообщает о них.

-создает объектный файл, который работает с относительными адресами ( неизвестна точка старта).

-после загрузки работает с абсолютными адресами (.exe). Для этого требуется загрузчик, который является частью операционной системы и компилятора.

После компиляции (экзекуции) информация приходит к драйверам- вывод на экран, принтер, остальную периферию.

Компиляторы бывают однопроходными, т.е. выполняют compile и linking за один раз ( присущи огромные размеры), и с разделенными функциями.

Для маломощных машин и программ, не зависящих от аппаратуры, созданы интерпретаторы. Для работы этих программ требуется небольшой объем памяти, т.к. интерпретатор выполняет инструкции последовательно, одна за другой, переводит текст и находит ошибки поштучно(первые версии Basic).

Современные Java – интерпретаторы, поэтому программы, написанные на этом языке, идут на любой маломощной машине, но работают они довольно медленно.