**Теория пределов.**

***Предел последовательности.***

Определение: Последовательностью называется любой счетный набор действительных чисел.

Определение: Числовая последовательность называется сходящейся, если . Число a называется пределом последовательности . Последовательность называется сходящейся к числу a, если . Всякая сходящаяся последовательность может быть представлена в виде , где – бесконечно малая последовательность.

***Бесконечно малая и бесконечно большая величины.***

Определение: Последовательность называется бесконечно большой, если .

Определение: Последовательность называется бесконечно малой, если .

***Арифметические действия с переменными, имеющими предел.***

Теорема: Если , а , то .

Доказательство: Пусть и и , где – бесконечно малая последовательность.

Теорема: Если , а , то .

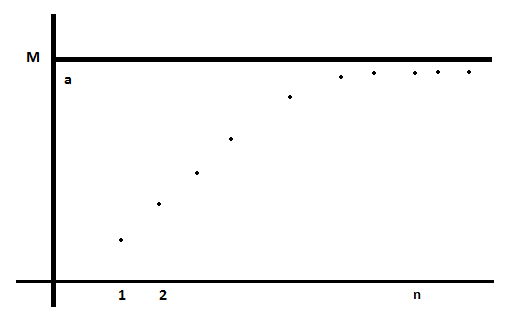
Доказательство: Пусть и и , где – бесконечно малая последовательность.

Теорема: Если , а , , то .

Доказательство: .

***Теорема о монотонной последовательности.***

Теорема Вейерштрасса (о сходимости монотонной последовательности): Если монотонно не убывает (не возрастает) и ограничена сверху (снизу), то она сходится (имеет конечный предел). Если не убывает, т.е. , то, очевидно, она ограничена снизу числом (если не возрастает, ограничена сверху числом ).

Доказательство:

Ограничимся случаем неубывающей последовательности (для невозрастающей аналогично). Пусть ограничена сверху существует точная верхняя грань .

По основному свойству точной грани: . Т.к. выполнено неравенство .

Итак: , т.е. .

***Второй замечательный предел.***

Теорема: – неопределенность 1∞.

Доказательство: Достаточно показать, что: .

1. При x → +∞ (в любом случае x положителен). Пусть x > 0, более того, x > 1. Тогда x = [x] + {x}. Обозначим [x] = n – натуральное число, а {x} = α.

.

Т.к. при x → +∞ и n → +∞ из того, что , . По теореме о предельном переходе в неравенствах получим требуемое.

1. При x → -∞. Пусть x < 0. . Доказано.

***Фундаментальная последовательность.***

Определение: Последовательность называется фундаментальной, если

*.*

***Критерий Коши сходимости последовательности.***

Теорема: Последовательность сходится она фундаментальна.

Доказательство:

Необходимость. Пусть сходится, т.е. . В частности . Т.к. . Получим: .

Достаточность. Пусть фундаментальна по лемме ограничена по теореме Больцано-Вейерштрасса из последовательности можно извлечь сходящуюся подпоследовательность: . Покажем, что . Пусть ε > 0 – произвольное число

1. Т.к. , то .
2. Из определения фундаментальности следует: .

Обозначим .

***Определения предела функции.***

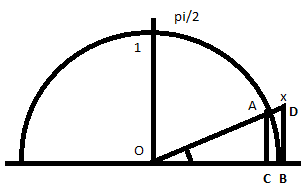
Определение (по Коши): y = f(x) имеет конечный предел в точке , равный A, если .

Определение (по Гейне): Число A называется пределом функции f(x) в точке , если . – соответствующая числовая последовательность.

***Первый замечательный предел.***

Теорема: – неопределенность .

Доказательство: Достаточно показать, что: .

1. При x→0+0. Рассмотрим .

Заметим, что площади ΔODB, ΔOAB и сектор OAB связаны соотношением:

;; .

.

, т.к. по теореме о предельном переходе в неравенствах получим требуемое.

1. При x→0-0 (т.к. функция четна) . Доказано.

***Непрерывность функции.***

Определение: Функция y = f(x) называется непрерывной в точке , если:

1. f(x) определена в точке ;
2. Существует конечный предел;
3. .

Определение: f(x) называется непрерывной в точке справа (слева), если: . Из теоремы об односторонних пределах функции: f(x) называется непрерывной в точке f(x) непрерывна в точке слева и справа.

***Разрывы первого и второго рода.***

Определение: Точка называется точкой разрыва функции y = f(x), если f(x) не является непрерывной в этой точке.

Определение: Точка называется точкой разрыва первого рода функции y = f(x), если односторонние пределы: и конечны, но не равны между собой. Величина называется скачком f(x) в точке .

Определение: Точка называется точкой разрыва второго рода, если хотя бы один из односторонних пределов либо бесконечен, либо не существует.

***Операции над непрерывными функциями.***

Теорема: Если f(x) и g(x) непрерывны в точке , то в этой точке непрерывны следующие функции: .

Доказательство: Докажем для , используя определение непрерывности по Гейне.

Пусть – произвольная. .

**Дифференциальное исчисление функции одной переменной.**

***Теоремы Вейерштрасса о функции непрерывной на отрезке.***

Теорема (первая теорема Вейерштрасса): Функция, непрерывная на отрезке, ограничена на нем.

Доказательство: Пусть функция y = f(x) определена и непрерывна на отрезке [a; b]. Предположим противное и функция f(x) не ограничена на отрезке [a; b]. Применим метод половинного деления отрезка. Разобьем [a; b] точкой на два отрезка [a; c] и [c; b]. Т.к. f(x) не ограничена на [a; b], то она не ограничена хотя бы на одном из отрезков (обозначим , длина ). Разобьем на два равных отрезка, причем выберем один из них, на котором функция не ограничена (обозначим , длина ). Продолжая процесс до бесконечности получим систему вложенных отрезков: Длина . По определению вложенных отрезков . Причем . Т.к. f(x) непрерывна на [a; b], то она непрерывна и в точке c (определение непрерывности в точке по Гейне). Причем по лемме о локальной ограниченности f(x) ограничена на . . . Если взять , . Итак, в f(x) ограничена, а в её подмножестве не ограничена. Получим противоречие.

Теорема (вторая теорема Вейерштрасса): Функция, непрерывная на отрезке, достигает на нем своих точных граней.

Доказательство: Пусть f(x) непрерывна на отрезке [a; b] по первой теореме Вейерштрасса . Не теряя общности докажем, что f(x) достигает на [a; b] своей точной верхней грани: . Предположим противное: . Рассмотрим вспомогательную функцию . непрерывна на (объединение всех граней для f(x)). Т.к. M – наименьшая из верхних граней, получим противоречие.

***Определение производной функции.***

Определение: Если существует конечный предел при Δx → 0 (Δx – приращение аргумента ) разностного отношения (Δy – приращение функции f(x)), то его называют производной функции f(x) в точке и обозначают .

***Геометрический и физический смысл производной функции.***

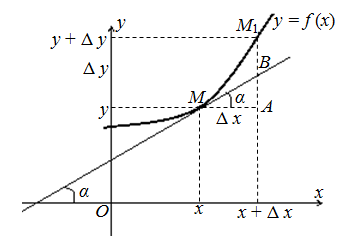
Обозначим следующие точки: A(, B(, C(). Если рассмотреть прямоугольный треугольник ABC, то можно заметить, что есть tg(BAC). При Δx → 0 точка B будет приближаться к точке A и секущая AB «превратится» в касательную графика f(x) в точке A. Обозначим за γ угол между этой касательной и положительным направлением оси Ox. Предельным переходом получим . Т.о. геометрический смысл производной: производная f(x) в точке равна тангенсу угла наклона касательной к графику функции f(x), проходящей через точку A. Физический смысл производной: если точка движется вдоль оси x и её координаты изменяются по закону x(t), то мгновенная скорость точки , а ускорение .

***Дифференциал функции.***

Определение: Функция y = f(x), определенная в некоторой окрестности точки , называется дифференцируемой в этой точке, если приращение Δy функции f(x) в точке , отвечающее приращению Δx аргумента , можно представить в виде: , где A – вещественное число, определяемое значением , – бесконечно малая функция при Δx → 0.

Определение: Главная линейная часть приращения Δy функции y = f(x) в точке , отвечающая приращению Δx аргумента , называется дифференциалом функции f(x) в точке .

– главная линейная часть приращения. .

***Геометрический смысл.***

Рассмотрим функцию f(x). Проведем к графику этой функции касательную MB в точке M и рассмотрим ординату этой касательной для точки . Рассмотрим прямоугольный ΔMAB. Тогда . Согласно геометрическому смыслу производной, тангенс угла между касательной к графику функции f(x) и положительным направлением оси абсцисс равен производной этой функции: . Следовательно, AB. Известно, что приращение независимой переменной равно дифференциалу этой переменной: Δx = dx. Тогда AB. По определению, дифференциал dy функции f(x) равен . Сравнивая правые части последних двух равенств, делаем вывод, что равны и их левые части, то есть AB = dy. Геометрический смысл дифференциала: дифференциал функции f(x) в точке есть приращение касательной в точке , отвечающее приращению Δx аргумента .

***Производная и дифференциал высшего порядка.***

Определение: Пусть на промежутке (a; b) определена производная функции f(x). Если – дифференцируемая на (a; b) функция, то её производная называется второй производной функции f(x) на промежутке (a; b) и обозначается . Аналогично определяется производная n-ого порядка от f(x): . Функция f(x) называется n-раз дифференцируемой на (a; b).

Определение: Пусть f(x) дважды дифференцируема на (a; b), тогда дифференциалом второго порядка функции f(x) на (a; b) называется выражение вида: . В общем случае определяется дифференциал n-ого порядка от функции f(x): .

***Теорема Коши о среднем.***

Теорема: Если функции f(x) и g(x) непрерывны на [a; b], дифференцируемы на (a; b), причем g(a) g(b), то .

Доказательство: Рассмотрим вспомогательную функцию: . F(x) удовлетворяет трём условиям теоремы Ролля:

1. F(x) непрерывна на [a; b] т.к. f(x) и g(x) непрерывны на [a; b].
2. F(x) дифференцируема на (a; b) т.к. f(x) и g(x) дифференцируемы на (a; b).
3. F(a) = 0 = F(b).

*,* т.е. .

***Теорема Лагранжа о среднем.***

Теорема: Если функция f(x) непрерывна на [a; b] и дифференцируема на (a; b), то .

Доказательство: Рассмотрим вспомогательную функцию . Она удовлетворяет теореме Ролля:

1. F(x) непрерывна на [a; b] т.к. f(x) непрерывна на [a; b].
2. F(x) дифференцируема на (a; b) т.к. f(x) дифференцируема на (a; b).
3. F(a) = 0 = F(b).

, т.е. .

***Формула Тейлора для функции с остаточным членом в форме Лагранжа.***

Теорема: Пусть функция f(x) дифференцируема в окрестности точки (n + 1)-раз, тогда для любого x из указанной окрестности выполняется равенство: . Эта формула называется не локальной формулой Тейлора, где. .

Доказательство: Рассмотрим вспомогательную функцию . Рассмотрим отношение – остаток формулы Тейлора в форме Лагранжа. Данная формула применяется при приближенных вычислениях, позволяет получать значение функции в заданных точках с любой наперед заданной точностью.

***Локальный экстремум функции одной переменной.***

Пусть функция f(x) определена на множестве .

Определение: Точка называется точкой строгого локального минимума (максимума) функции, если .

Если в определении неравенства нестрогие, то называется точкой нестрогого локального минимума (максимума). Все такие точки называются точками локального экстремума.

***Необходимые условия экстремума функции.***

Теорема: Если функция f(x) дифференцируема в точке c и точка c – точка локального экстремума f(x), то или не существует.

Доказательство: Следует из теоремы Ферма.

***Достаточные условия экстремума функции одной переменной.***

Теорема (первое достаточное условие экстремума): Пусть точка c – это точка возможного экстремума функции f(x), дифференцируемой в некоторой окрестности точки c. Если

в некоторой левой полуокрестности точки c и в некоторой правой полуокрестности точки c, то точка c – точка локального максимума (минимума) функции f(x).

Доказательство: Ограничимся случаем, когда в окрестности меняет знак при переходе через точку c с плюса на минус. Возьмем произвольную точку . c – точка локального максимума.

Теорема (второе достаточное условие экстремума): Пусть и . Если , то c – точка локального экстремума функции f(x), причем, если , то – точка локального минимума, если , то c – точка локального максимума.

Доказательство: возрастает в точке c справа от c и слева от c по первому достаточному условию получаем, что c – точка локального минимума.

Теорема (третье достаточное условие экстремума): Пусть n – натуральное число. Функция f(x) дифференцируема в окрестности точки c (n + 1)-раз, – непрерывна в точке c. Тогда, если , то при (n + 1) четном точка c – точка экстремума функции f(x) (максимум при и минимум при ).

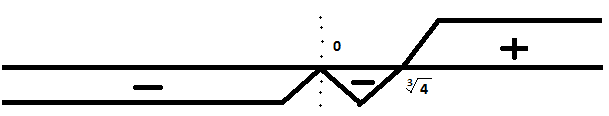
***Исследование функции.***

Исследование функции f(x) проводится обычно по следующей схеме:

1. Находим область определения D(f) и область значения E(f).
2. Исследование на чётность/нечётность, периодичность/не периодичность.
3. Нахождение точек пересечения графика f(x) с осями координат, нахождение промежутков знакопостоянства функции.
4. Асимптоты графика.
5. Использование , нахождение промежутка монотонности и точек экстремума.
6. Использование , нахождение промежутка выпуклости и точек перегиба.
7. Построение графика.

***Построение графика функции.***

Построить график .

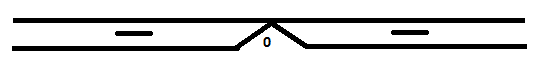
1. .
2. функция ни чётная, ни нечётная.
3. С осью Oy нет точек пересечения (). С осью .

Промежутки знакопостоянства: .

1. Вертикальные: . Наклонная: наклонная асимптота при .
2. .

.

1. .

Функция выпукла вверх.

**Интегральное исчисление функции одной переменной.**

***Первообразная и неопределенный интеграл.***

Определение: Функция F(x) называется первообразной для функции f(x) на , если .

Определение: Множество всех первообразных для функции f(x) на промежутке X называется неопределенным интегралом от функции f(x) на X и обозначается: , где F(x) – одна из первообразных функции f(x) на X, C – произвольная постоянная.

***Таблица основных интегралов.***

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

***Простейшие правила интегрирования.***

1. Если a – постоянная (, то .

Действительно, дифференцируя выражение справа, получим так что выражение является первообразной для дифференциала .

1. .

Дифференцируем выражение справа: . Т.о., это выражение является первообразной функцией для последнего дифференциала.

1. Если , то .

Действительно, данное соотношение равносильно следующему: . Но тогда , так что . Т.е. действительно оказывается первообразной для функции .

***Интегрирование путем замены переменной.***

Теорема: Если функция дифференцируема на множестве T и принимает значение на множестве , а функция f(x) интегрируема на множестве X, то справедливо равенство:

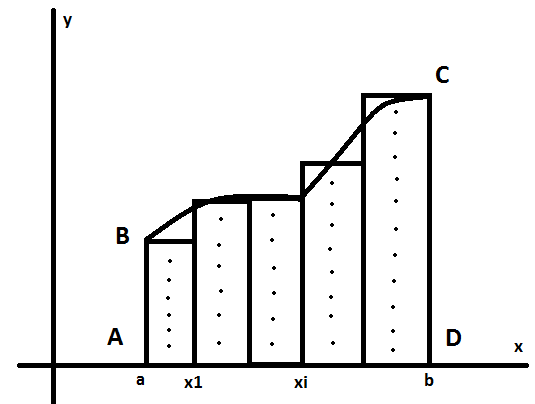
Доказательство: Пусть F(x) – первообразная для f(x) на X. Положив, что получим: . Т.к. получим, что левая часть доказываемого равенства – семейство функций , а правая часть – это семейство функций слева и справа одно и тоже семейство множеств.

Применение данной формулы слева направо называется методом внесения под знак дифференциала, а справа налево – методом подстановки.

***Интегрирование по частям.***

Теорема: Пусть функции u(x) и v(x) дифференцируемы на множестве X. Если функция интегрируема на X, то существует первообразная и для функции и справедлива формула: . Эта формула позволяет вычислить интеграл в левой части равенства, если интеграл в правой части вычисляется легче.

Доказательство: Воспользуемся формулой производной произведения. . Проинтегрируем обе части: .

***Понятие определенного интеграла, его геометрический смысл.***

Пусть функция y = f(x) определена на [a; b]. Рассмотрим разбиение отрезка [a; b] точками . На каждом элементарном отрезке выберем точку . Рассмотрим следующую конечную сумму: , где – длина отрезка . Каждое слагаемое суммы – площадь заштрихованного прямоугольника с основанием и высотой . Сумму называют интегральной суммой для функции f(x), отвечающей разбиению и набору и геометрически выражает площадь заштрихованного многоугольника. Обозначим .

Определение: Если существует конечный предел последовательности интегральных сумм при , не зависящий от способа разбиения и от выбора точек , то его (предел) называют определённым интегралом от функции f(x) по промежутку [a; b] и обозначают . Функция f(x) называется интегрируемой на отрезке [a; b]. Геометрически выражает предел последовательности площадей системы прямоугольников и равен площади криволинейной трапеции ABCD, ограничиваемой линиями x = a, x = b, y = 0, y = f(x).

***Свойства определенных интегралов.***

1. .

Доказательство: .

1. Если f(x) интегрируема на [a; b], то .

Доказательство: Рассмотрим разбиение отрезка [a; b] на элементарные отрезки . При этом разбиение справа налево: . Т.к. , то .

1. Если f(x) и g(x) интегрируемы на [a; b], то и функция f(x) + g(x) также интегрируема на [a; b], причем .

Доказательство: Рассмотрим разбиение , составляющее интегральные суммы, получим: существует предел левой части равенства, являющийся интегральной суммой для интеграла , к которому стремится при .

1. Если f(x) интегрируема на [a; b], то функция также интегрируема на [a; b], причем .

Доказательство: Получается предельным переходом в равенстве: .

1. Если f(x) интегрируема на [a; b], то f(x) – интегрируема на и выполняется равенство .

Доказательство: Будем брать такие разбиения отрезка [a; b], что , тогда , где и соответствуют разбиениям отрезков [a; c] и [c; b] соответственно. Т.к. Аналогично из неравенства . В равенстве переходим к пределу при и получаем требуемое.

***Интеграл как функция верхнего предела.***

Пусть функция y = f(x) интегрируема на [a; b] по свойству 5. существует интеграл определена функция аргумента x , называемая определенным интегралом с переменным верхним пределом.

***Формула Ньютона-Лейбница.***

Согласно теореме (если f(x) непрерывна на [a; b], то F(x) дифференцируема на (a; b)), – первообразная для – множество всех первообразных для функции f(x).

Т.к. и . Итак , где – любая из первообразных для f(x).

***Суммы Дарбу, условия существования интеграла.***

Пусть функция y = f(x) определена и ограничена на [a; b]. Рассмотрим разбиение отрезка [a; b]. Обозначим – точная нижняя грань и – точная верхняя грань.

Определение: Конечные суммы и называются нижними и верхними суммами Дарбу функции f(x), отвечающим разбиению на отрезке [a; b].

Геометрически нижняя сумма Дарбу выражает площадь системы прямоугольников, покрываемой криволинейной трапецией ABCD, а верхняя сумма Дарбу – площадь системы прямоугольников, покрывающей трапецию ABCD.

Заметим, что .

Теорема (необходимое и достаточное условие интегрируемости функции): Интеграл существует .

***Интегрируемость непрерывных и монотонных функций.***

Теорема: Если f(x) непрерывна на отрезке [a; b], то f(x) интегрируема на [a; b].

Доказательство: Заметим, что функция f(x) на [a; b] по критерию интегрируемости: . Т.к. f(x) непрерывна на отрезке [a; b], то f(x) равномерно непрерывна на [a; b] . Возьмем λ достаточно малым, а именно λ < δ. Будем рассматривать такие разбиения , что . В этом случае: .

Теорема: Функция, монотонная на [a; b], интегрируема на [a; b].

Доказательство: Пусть, например, функция f(x) монотонно не убывает на [a; b]. Тогда .

***Несобственные интегралы.***

Определение: Пусть функция y = f(x) определена на луче и интегрируема на отрезке . Если существует конечный предел , то его называют несобственным интегралом от f(x) по промежутку и обозначают: . Функция f(x) в этом случае называется интегрируемой по в несобственном смысле. Если предел не существует или бесконечен, то интеграл называется расходящимся. Геометрически интеграл выражает площадь неограниченной фигуры, заключенной между линиями x = a, y = 0, y = f(x).

Определение: Если функция f(x) определена на множестве , интегрируема на и существует конечный предел , то его значение – это несобственный интеграл .

Определение: Если f(x) определена на , то несобственным интегралом от f(x) по промежутку называется выражение: , где – произвольное. Заметим, что интеграл в левой части сходится сходятся оба интеграла в правой части.

***Несобственные интеграл от неотрицательных функций.***

Если f(x) > 0 на луче , то функция является неубывающей на . Воспользуемся следующей теоремой (аналог теоремы Вейерштрасса): Если f(x) монотонна на и ограничена, то существует конечный предел .

Отсюда следует следующая теорема:

Теорема: Интеграл сходится .

Доказательство: Очевидно – монотонна, непрерывна и ограничена сверху .

***Признаки сходимости.***

Теорема (признак сравнения): Если на луче определены неотрицательные функции f(x) и g(x), интегрируемые на любом отрезке и , то

1. из сходимости сходимость ;
2. из расходимости расходимость .

Доказательство:

1. Пусть сходится . сходится.
2. Предположим противное. Пусть сходится сходится сходится.

Теорема (предельный признак сравнения): Если на луче определены положительные функции f(x) и g(x) и существует конечный предел . Функции f(x) и g(x) интегрируемы на , то интегралы и сходятся или расходятся одновременно.

**Дифференциальное исчисление функции многих переменных.**

***Частная производная функции многих переменных.***

Пусть функция определена на множестве . Пусть точка – внутренняя точка множества D. Придадим координате приращение так, что точка также принадлежит D. Рассмотрим соответствующее частное приращение функции f(x) в точке M: .

Определение: Если существует конечный предел разностного отношения при , то его называют частной производной функции f(x) в точке M по переменной и обозначают или . На практике при вычислении частной производной по остальные переменные принимаются равными константе.

***Производная по направлению функции многих переменных.***

Пусть функция u = f(x,y,z) определена в окрестности точки . Рассмотрим произвольный единичный вектор , где . Рассмотрим прямую в : .

Определение: Производной функции в точке по направлению называется производная функции по l при l = 0 и обозначается . По формуле дифференцирования сложной функции получим: .

***Градиент функции многих переменных, его свойства.***

Определение: Градиентом функции называется выражение . Формула для производной по направлению примет вид: .

Докажем, что направление градиента – это направление наискорейшего роста функции f(x,y,z).

Действительно, , причем равенство достигается при , т.е. наискорейший рост функция многих переменных испытывает в направлении градиента. Из сказанного следует, что градиент функции можно определить как вектор, обладающий следующими двумя свойствами:

1. длина его равна максимальной величине производной по направлению (для дифференцируемой в (x, y, z) функции этот максимум существует и есть число неотрицательное);
2. если его длина не равна нулю, то он направлен в ту же сторону, что и вектор l, вдоль которого производная максимальна.

***Полный дифференциал функции многих переменных.***

Определение: Дифференциалом функции f(x) в точке , отвечающем приращением аргумента , называется главная линейная часть приращения функции f(x) в точке M, если f(x) – дифференцируема в точке M и обозначается . Заметим, что для дифференцируемой функции в точке M получается расчетная формула для дифференциала: где . Если – независимые переменные, то справедлива формула: . Если же , то для сложной функции от переменных : .

***Дифференциалы высших порядков.***

Определение: Дифференциалом второго порядка от функции f(x,y) называется выражение вида: . В случае, когда x и y – независимые переменные, . Поэтому: . Аналогично получается для функции m переменных: . Аналогично получается для дифференциалов любого порядка, а именно : .

***Экстремумы функции многих переменных.***

Пусть определена в области D, точка – внутренняя точка области.

Определение: Если существует такая выколотая окрестность , что , то точка называется точкой строгого локального минимума (максимума). Если в определении неравенства нестрогие, то точка – точка нестрогого локального минимума (максимума).

***Необходимые условия.***

Теорема. Пусть точка – точка локального экстремума функции . Если существуют частные производные функции f(x) в точке , то выполняется: , ..

Доказательство: Пусть, например, – точка локального максимума функции . Рассмотрим произвольный . Зафиксируем все координаты, кроме i-той: . Получим функцию: – функция одной переменной. Т.к. . Действительно, . Т.к. . точка локального максимума функции .. Т.к. .

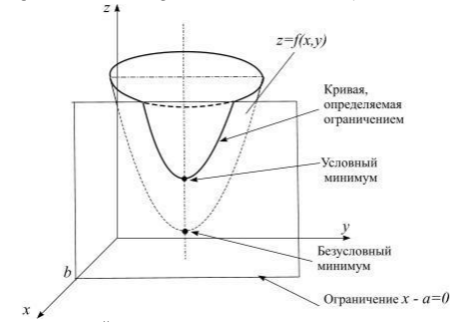
***Достаточные условия экстремума функции многих переменных.***

Теорема. Пусть дифференцируема в окрестности точки , частные производные второго порядка непрерывны в точке и точка является стационарной точкой функции u = f(x). Тогда, если , то точка – точка локального минимума (максимума).

Доказательство: Ограничимся случаем . Обозначим , где . Т.к. . Поэтому условие равносильно условию знакоположительной (знакоотрицательной) квадратичной формы: .

Итак, переформулируем теорему: Если в стационарной точке функции выполнены условия , то – точка локального экстремума, причем, если , то – точка локального минимума, если , то – точка локального максимума. Если же , то точка не является точкой локального экстремума.

***Понятие условного экстремума.***

Пусть дано уравнение ϕ(x, y) = 0 и точка удовлетворяет этому уравнению. Пусть функция z = f(x, y) определена в некоторой окрестности точки и непрерывна в этой точке. Тогда если для всех точек (x, y) этой окрестности, удовлетворяющих уравнению ϕ(x, y) = 0, выполняется неравенство (), то называется точкой условного максимума (условного минимума) функции z = f(x, y) , а ϕ(x, y) = 0 – уравнением связи. Уравнение ϕ(x, y) = 0, задающее некоторую кривую на плоскости xy, определяет в пространстве xyz цилиндрическую поверхность, образующая которой параллельна оси z. Пусть функция z = f(x, y) определяет некоторую поверхность в пространстве xyz и цилиндрическая поверхность ϕ(x, y) = 0 высекает из поверхности z = f(x, y) некоторую линию. По форме этой линии можно судить об условных максимумах и минимумах функции z = f(x, y) .

***Метод множителей Лагранжа.***

Метод множителей Лагранжа поиска условного экстремума рассмотрим для случая функции двух переменных z = f (x, y) и одного уравнения связи ϕ(x, y) = 0. Рассмотрим функцию L(x, y, λ) = f(x, y)+λϕ(x, y). Число λ называется множителем Лагранжа, а функция L(x, y, λ) функцией Лагранжа. Метод множителей Лагранжа применяется при определенных ограничениях на функции f(x, y) и ϕ(x, y) и вытекает из двух теорем.

Теорема: Пусть – точка условного экстремума функции f(x, y) при наличии связи ϕ(x, y) = 0, и пусть функции f(x, y) и ϕ(x, y) непрерывно дифференцируемы в окрестности точки и хотя бы одна из частых производных , отлична от нуля. Тогда найдется такое значение , что удовлетворяет системе , , . Заметим, что точка , определяемая системой, называется стационарной точкой.

Теорема (достаточные условия условного экстремума): Пусть функции, f(x, y) и ϕ(x, y) имеют непрерывные частные производные второго порядка в окрестности точки , стационарная точка функции Лагранжа L(x, y, λ) и Δ – определитель вида . Тогда, если Δ < 0 , то функция z = f(x, y) имеет в точке условный максимум; если Δ > 0, то условный минимум. Наличие ограничений на функции f(x, y) и ϕ(x, y) в формулировках теорем говорит о том, что есть функции z = f(x, y) и ограничения ϕ(x, y), для которых точки условного экстремума могут не быть стационарными точками.

**Числовые ряды. Степенные ряды.**

***Числовые ряды, основные понятия, сходимость ряда.***

Рассмотрим числовую последовательность . Бесконечная сумма вида . называется числовым рядом, обозначается . Слагаемое называется n-ым числом ряда.

Определение: Конечная сумма называется n-ой частичной суммой числового ряда.

Определение: Если существует конечный предел последовательности частичных сумм , то его называют суммой числового ряда, а сам ряд называют сходящимся к сумме S. Если же предел не существует либо бесконечен, то числовой ряд называется расходящимся.

Заметим, что между элементами последовательностей и можно установить взаимно-однозначное соответствие: . С другой стороны . Т.е. .

Определение: Бесконечная сумма называется n-ым остатком ряда.

Лемма: Если ряд сходится, то .

Доказательство: Пусть ряд сходится .

Лемма (необходимый признак сходимости числового ряда):Если ряд сходится, то .

Доказательство: Пусть ряд сходится: .

***Абсолютная и условная сходимость рядов.***

Определение: Если для степенного ряда сходится ряд из модулей его членов , то степенной ряд называется абсолютно сходящимся.

Теорема: Из абсолютной сходимости ряда следует сходимость ряда .

Доказательство: Пусть сходится абсолютно сходится знакоположительный ряд |Критерий Коши| . Обратное в общем случае неверно.

Определение: Если числовой ряд сходится, но не сходится абсолютно, то его называют не абсолютно-сходящимся (условно сходящимся).

***Ряды с неотрицательными членами.***

Будем исследовать знакоположительные числовые ряды . Заметим, что если числовой ряд знакоположительный, то – монотонно не убывает. Поэтому, применяя теорему о сходимости монотонной последовательности, получим:

Теорема: Знакоположительный ряд имеет конечную сумму .

Замечание: Из теоремы следует, что знакоположительный ряд имеет бесконечную сумму – не ограничена сверху.

***Признаки сходимости положительных рядов.***

Теорема (признак сравнения): Пусть и – знакоположительные ряды. Пусть . Тогда:

1. Из сходимости ряда следует сходимость .
2. Из расходимости ряда следует расходимость .

Доказательство: Т.к. при отбрасывании конечного числа членовсходимость сохраняется, будем считать, что .

1. Пусть сходится ряд ряд сходится.
2. Пусть расходится ряд . Предположим противное. Пусть ряд сходится – сходится (противоречие).

Теорема (предельный признак сравнения): Если существует конечный предел , то знакоположительные ряды и сходятся или расходятся одновременно.

Доказательство: Пусть , т.е. . Пусть сходится , тогда сходится и ряд сходится и ряд . Пусть сходится ряд сходится ряд сходится . Т.о. сходится сходится.

***Степенные ряды.***

Определение: Функциональный ряд вида , где – постоянные, называется степенным рядом. При ряд примет вид: . Областью сходимости степенного ряда называется множество тех значений х, при которых степенной ряд сходится.

***Промежуток сходимости степенного ряда.***

Теорема Абеля: Если степенной ряд сходится в точке , то он сходится абсолютно в интервале и равномерно на любом отрезке [-q, q], где .

Из теоремы Абеля следует, что, если степенной ряд сходится в точке и расходится в точке , то , что ряд сходится абсолютно в интервале , расходится в , в точке требуются дополнительное исследование. Число R называется радиусом сходимости ряда, а интервал – его интервалом сходимости.

Очевидно, что в точке x = 0 ряд сходится. Если ряд сходится только в точке x = 0, то R = 0. Если ряд сходится , то . Для нахождения радиуса сходимости ряда используют признаки Коши и Даламбера:

1. Пусть . Ряд сходится в точке x абсолютно, если выполняется условие (признак Коши): , т.о .
2. Пусть . Тогда ряд сходится в точке x абсолютно, если выполняется условие (признак Даламбера): .

**Типы данных.**

***Понятие типа данных в различных языках программирования.***

Тип – это некоторое множество значений и набор операций, которые могут выполняться над этими значениями. Типы могут классифицироваться по критериям скалярности и предопределенности. Скалярные типы подразделяются на скалярные и структурированные.

Скалярный тип – это тип, значение которого скалярное, т.е. не имеет составных частей.

Структурированный тип – это тип, значение которого состоит из элементов, имеющих самостоятельный смысл.

Предопределенный тип – это тип, значение которого существует изначально и для его использования не требуется дополнительных усилий со стороны программиста.

Другие типы создаются программистом самостоятельно с помощью специальных языковых конструкций и называются конструированными.

***Основные типы (на примере языка Паскаль или др.).***

В Pascal существуют следующие скалярные типы:

1. Числовые – целочисленные и вещественные типы. Значениями этих типов являются целые и вещественные числа.
2. Символьный тип. Значением типа являются отдельные символы, заключенные в апострофы.
3. Логический, булевский тип. Принимает два значения: истина (true) или ложь (false).
4. Адресный тип. Значением типа являются адреса ячеек памяти.

Существуют следующие структурированные типы:

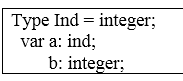
1. Строковый тип. Значением типа является последовательность символов. С ними можно работать как с единым целым, так и с отдельным символом строки.
2. Регулярный тип. Значением является массив. Массив представляет собой набор однотипных элементов, количество и тип которых не могут изменяться в течение работы программы.
3. Комбинированный тип. Значением типа являются записи. Запись – это совокупность возможно разнотипных элементов, количество которых фиксировано. Элемент записи называется полем.
4. Множественный тип. Значением типа являются множества, подчиняющиеся теоретико-множественным операциям (объединение, пересечение и т.д.).
5. Файловый тип. Значением типа являются файлы, которые представляют последовательность записей, хранимых на внешних носителях.
6. Функциональные и процедурные типы. В качестве типа используется класс подпрограмм, имеющих общий способ вызова.
7. Объектный тип. Значением типа являются объекты, содержащие данные, называемые полями, и процедуры и функции, называемые методами, которые обрабатывают эти данные.

Язык Pascal представляет возможность конструировать типы:

1. Интервальный тип. Представляет диапазон некоторого скалярного типа, который может быть как предопределенным, так и сконструирован программистом.
2. Перечислимый тип. Представлен в виде перечня имен, констант определенного типа.
3. Ссылочный тип. Значением типа являются адреса ячеек памяти, но каждая переменная содержит дополнительную информацию, позволяющую корректно обращаться к ссылочным значениям.
4. Структурированный тип. Структурированные типы данных также могут быть конструируемыми.

***Конструирование новых типов данных.***

Конструируемые типы определяются в программе по мере необходимости. Языковая конструкция, с помощью которой определяется новый тип, называется конструктором типа или определением типа. Для каждого класса типов конструктор имеет определенную структуру. В определении типа программист задает параметры, которые определяют свойства типа. Языковая конструкция, с помощью которой программист ставит в соответствие имени типа его определение, называется описанием типа.

Описание типа имеет следующую структуру: Type <имя типа> = <определение типа>;

Если конструированный тип не поименован, т.е. нет соответствующего описания, то такой тип называется безымянным или анонимным (var <имя переменной>:<тип>;). В качестве типа можно использовать либо имя типа, как определенное, так и конструированное, либо определение типа. Т.к. язык Pascal является строго типизированным языком, то анонимный тип не может совпадать ни с каким другим типом.

***Структурные типы данных в языках программирования.***

Пусть – некоторое множество переменных. В общем случае это множество может быть бесконечным. И пусть – множество отношений, заданных на множестве D. Тогда будем называть информационной структурой или структурой данных. Элементы множества D называются узлами информационной структуры. Подразумевается, что каждый узел имеет вполне определенный тип. Важнейшей классификацией информационных структур являются линейные информационные структуры, т.е. структуры, на узлах которых определены отношения линейного порядка. Для таких структур используется понятие первого и последнего узла. Для любых информационных структур определены операции, например, операции доступа к узлу структуры (в массиве доступ осуществляется по индексу). Различают статические (массив) и динамические (файлы) информационные структуры. В статических структурах количество узлов и их состав не изменяются во время существования структуры, в отличие от динамических. Следует различать логическую организацию информационной структуры и её физическую интерпретацию конкретными средствами языков программирования.

Различают следующие линейно-динамические информационные структуры:

1. Стек. Доступ к элементам осуществляется по принципу LIFO (последний пришел – первый вышел). Элемент или узел, помещенный в стек последним, называется верхушкой стека. Элемент или узел, помещенный в стек первым, называется дном стека. Над стеком выполняются операции:
2. Добавить узел (только на верхушку).
3. Извлечь узел (только с верхушки).
4. Инициализация стека (создание пустого стека).
5. Стек пуст (булевская функция, определяющая пустоту стека).
6. Очередь. Доступ к элементам осуществляется по принципу FIFO (первый пришел – первый вышел). Над очередью выполняются операции:
7. Добавить узел (только на верхушку).
8. Извлечь узел (только с дна).
9. Инициализация стека (создание пустой очереди).
10. Очередь пуста (булевская функция, определяющая пустоту очереди).
11. Дек. Имеет два входа и выхода (двусторонняя очередь).

***Особенности их представления в памяти ЭВМ.***

1. Последовательное распределение памяти. Это ситуация, когда логически-соседние узлы информационной структуры располагаются в физически-смежных блоках памяти.
2. Связное распределение памяти. Это ситуация, когда логически-соседние узлы информационной структуры размещаются не в смежных блоках физической памяти. При связном распределении узел информационной структуры, кроме информационной составляющей (значение узла), должен хранить информацию о местоположении следующего узла.

Основной недостаток связного распределения памяти состоит в обязательном выделении памяти под поле связи, однако эти затраты существенно не влияют на пространственную сложность алгоритма, т.к. в большинстве программ поле Info занимает значительно больше памяти, чем поле Next. Связное распределение памяти обладает основным достоинством перед последовательным, т.к. не требуется выполнение массовых операций.

**Файлы в языках программирования.**

***Файловые типы данных в языках программирования.***

1. Типизированные файлы представляют собой набор записей, имеющих одинаковую структуру, которая определятся в момент описания файла. Записи хранятся в машинном представлении, и виртуальная Паскаль-машина может контролировать корректность обращения к записям файла.

Описание типизированного файла имеет вид: var f: file of <тип записи>;

Тип записи может быть любым, кроме файлового или объектного.

1. В бестиповых файлах информация о записях хранится в машинном представлении, но о структуре записи ничего неизвестно. Т.о. программа может интерпретировать содержание файла по-разному. Виртуальная Паскаль-машина не отвечает за корректность доступа к записям. Описание бестипового файла имеет вид: var f:file;

Размер записи файла определяется в момент открытия файла в процедуре reset или rewrite. Вызовы этих процедур для бстиповых файлов имеют вид: reset/rewrite(f,bufsize)., где второй параметр задает размер записи в байтах. Ввод и вывод в бестиповом файле задается с помощью процедур blockread/blockwrite(f,d,size,[k]), где d – имя некоторой переменной, задающей начало буферной области, в которую будет записываться или считываться запись файла; size – параметр, задающий количество блоков, длины bufsize; k является входным параметром и по завершению работы содержит реальное количество записей, перемещаемых между файлом и основной памятью.

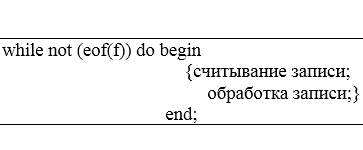
Типизированные и бестиповые файлы могут использоваться как файлы прямого доступа. Нумерация записей в файле начинается с нуля. Доступ к записи файла осуществляется по номеру. Под текущей записью понимается та запись, из которой выполняется чтение или в которую выполняется запись в данный момент.

1. Текстовые файлы хранят информацию в виде ASCII-кодировки. В отличие от предыдущих классов, записи текстовых файлов могут иметь различную длину. Каждая строка в текстовом файле оканчивается составным символом конца строки, состоящим из двух символов – «перевод строки» #10(LF) и «возврат каретки» #13(CR).

***Операции с файлами.***

Пусть f – имя файловой переменной, s – выражение строкового типа.

Над файлами выполняются следующие операции:

1. assign(f,s) – процедура связывания логического файла f с физическим файлом, имя которого задано выражением s. Имя физического файла задается по правилам операционной системы.
2. reset(f), rewrite(f) – процедура открытия файла, соответствующего переменной f. reset открывает существующий файл и устанавливает доступ к первой записи файла. rewrite создает файл и подготавливает доступ к записи: если файл существует, то он уничтожается и создается снова.
3. close(f) – закрытие файла. Процедура обеспечивает целостность данных.
4. eof(f) – булевская функция. Возвращает значение true, если достигнут конец файла.

Поскольку заранее количество записей в файле неизвестно, используется стандартная схема обработки записей файла.

1. rename(f,s) – процедура переименования файла, связанного с переменной f. Новое имя файла хранится в выражении s. К моменту переименования файл должен быть закрыт.
2. erase(f) – процедура удаления файла, связанная с переменной f. К моменту удаления файл должен быть закрыт.

Для файлов прямого доступа (типизированные и бестиповые):

1. filesize(f) – функция, возвращает количество записей от f.
2. filepos(f) – функция, возвращает номер текущей записи файла f.
3. seek(f,n) – устанавливает в качестве текущей запись с номером n из файла f.

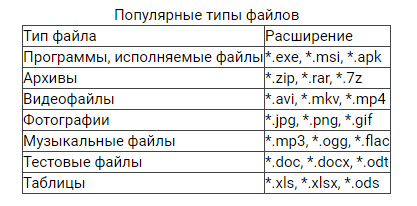
Для файлов последовательного доступа (текстовые):

1. append(f) – процедура, открывает текстовый файл для добавления в конец файла.
2. settextbuf(f,buf,size) – процедура, задает в явном виде буфер ввода/вывода для записи файла.
3. eoln(f) – булевская функция, возвращает true, если достигнут конец строки.
4. seekeof(f) – булевская функция, возвращает true, если достигнут конец файла или от текущей позиции до конца находятся только пробелы или символы конца строки.
5. seekeoln(f) – булевская функция, возвращает true, если достигнут конец строки или от текущей позиции до конца находятся только пробелы.

***Понятие формата файла.***

Формат — спецификация [структуры данных](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), записанных в компьютерном [файле](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB). Существует множество типов и форматов файлов. В общем виде формат – это сведения о том, как программа должна интерпретировать содержимое файла при его открытии. Иногда формат путают с расширением имени файла, но это совершенно разные понятия. Операционная система может использовать расширение имени файла, чтобы быстро определить, какой программой должен открываться данный файл. Как правило, расширение должно соответствовать формату, но это вовсе не обязательно. Сведения о формате находятся в самом файле – они помещаются в начало последовательности байтов.

***Примеры известных форматов файлов.***

Текстовый файл – самый простой формат. Каждый байт текстового файла является кодом определенного символа ASCII (буквы, цифры или знака препинания), а также символов пробела, начала и конца строки. Кроме значений текстовых символов, в текстовом файле действительно больше ничего нет! Структура любых других типов файлов гораздо сложнее. Наиболее популярные пользовательские форматы файлов можно сгруппировать по типу их использования и, следовательно, по расширению. Большинство указанных типов файлов открываются штатными средствами операционных систем, и работа с ними не требует от пользователя каких-либо специальных знаний и умений. С документами ситуация обстоит несколько по-другому. Для работы с ними уже требуются специальные пакеты программ.

**Работа со списками.**

***Работа со списками в языках программирования.***

Определение ссылочного типа имеет вид: ^<базовый тип>. Ссылочные значения не имеют внешнего представления, поэтому их нельзя вводить и выводить. Переменная ссылочного типа может быть получена в результате присваивания или вызова специальной встроенной процедуры new. Константа nil обозначает ссылку в никуда. С каждой ссылочной переменной связана переменная базового типа, которая обозначается как p^. Такие переменные не описываются в программе и называются указуемыми, т.е. в программе описываются переменные ссылочного типа, а далее появляется ссылочная переменная с помощью процедуры new(p). Затем указуемой переменной можно присваивать значение базового типа. Указуемую переменную в конце работы программы можно удалить вызовом специальной встроенной процедуры dispose(p). Рассмотрим последовательность вызовов: new(p); ; new(p); . В этом случае первая ячейка становится недоступной и называется мусором. Количество мусора может привести к некорректному завершению программы, т.е. вся свободная память будет исчерпана. Для устранения этой проблемы необходимо запоминать ссылки на выделенные ячейки памяти.

***Представление списков с использованием динамической (управляемой) памяти (на примере языка Паскаль или др.).***

Списком называется структура данных, каждый элемент которой посредством указателя связывается со следующим элементом. Каждый элемент связанного списка, во-первых, хранит какую-либо информацию, во-вторых, указывает на следующий за ним элемент. Так как элемент списка хранит разнотипные части (хранимая информация и указатель), то его естественно представить записью, в которой в одном поле располагается объект, а в другом – указатель на следующую запись такого же типа. Такая запись называется звеном, а структура из таких записей называется списком или цепочкой. Лишь на самый первый элемент списка (голову) имеется отдельный указатель. Последний элемент списка никуда не указывает (null). Построение однонаправленного списка в прямом порядке:

type list = ^elem;

elem = record

info: node;

next: list;

end;

var p, q, l: list; x: node;

begin new(l); p := l; (строим первый элемент)

readln(x); p^.info := x; (считывает с клавиатуры x и помещает в поле info)

while x <> 0 do begin (наполняем список)

readln(x);

if x <> 0 then begin

new(q);

q^.info := x;

p^.next := q;

p := q;

end; end;

p^.next := nil; (закрываем)

|  |  |
| --- | --- |
| Обработка списка | Вывод списка |
| p := l;  while p <> nil do begin  {обработка}  p := p^.next;  end; | p := l;  while p <> nil do begin  writeln(p^.info);  p := p^.next;  end; |

***Операции со списками.***

1. Удаление первого положительного элемента:

if l^.info > 0 then

l := l^.next;

1. Удаление положительных элементов:

p := l;

while p^.next <> nil do begin

if p^.next^.info > 0 then p^.next := p^.next^.next

end; p := p^.next;

end;

1. Вставка до каждого положительного:

|  |  |
| --- | --- |
| p := l;  while p^.next <> nil do begin  if p^.next^.info > 0 then begin  new(q); q^.info := x;  q^.next := p^.next; p^.next := q;  p := p^.next;  end;  p := p^.next;  end; |  |

1. Вставка после каждого положительного:

p := l;

while p <> nil do begin

if p^.info > 0 then begin

new(q); q^.info := x;

q^.next := p^.next; p^.next := q;

p := p^.next;

end;

p := p^.next;

end;

**Бинарные деревья, множества, графы и их обработка.**

***Работа с бинарными деревьями в языках программирования.***

Определение: Корневым двоичным деревом называют либо пустое множество, либо не пустое множество, в котором выделен один элемент t, называемый корнем дерева, а все остальные элементы разбиты на два непересекающихся подмножества и, каждое из которых в свою очередь является корневым двоичным деревом. Двоичное дерево называется двоичным деревом поиска, если:

* У всех узлов левого поддерева произвольного узла n значения ключей данных меньше, чем значение ключа данных самого узла n;
* У всех узлов правого поддерева произвольного узла n значения ключей данных не меньше, чем значение ключа данных узла n;
* Оба поддерева являются двоичными деревьями поиска.

Данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше. Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных.

Определение: Узлы, являющиеся корнями левого и правого поддеревьев, называются сыновьями. Узлы, не имеющие сыновей, называются листьями. Узел n является потомком узла k, если либо n является сыном узла k, либо существует такой узел l, который является сыном узла k и n является потомком узла l.

Для моделирования деревьев в Паскаль определим следующий тип:

type tree = ^elem;

elem = record

info: node;

l, r: tree;

end;

***Представление бинарных деревьев с использованием динамической (управляемой) памяти (на примере языка Паскаль или др.).***

Бинарные деревья достаточно просто могут быть представлены в виде списков или массивов. Списочное представление бинарных деревьев основано на элементах, соответствующих узлам дерева. Каждый элемент имеет поле данных и два поля указателей. Один указатель используется для связывания элемента с правым потомком, а другой с левым. Листья имеют пустые указатели потомков. При таком способе представления дерева обязательно следует сохранять указатель на узел, являющийся корнем дерева. Можно заметить, что такой способ представления имеет сходство с простыми линейными списками. И это сходство не случайно. На самом деле рассмотренный способ представления бинарного дерева является разновидностью мультисписка, образованного комбинацией множества линейных списков. Каждый линейный список объединяет узлы, входящие в путь от корня дерева к одному из листьев. Обозначим как root – корень дерева, а x – элемент, помещаемый в информационную часть нового узла.

|  |  |
| --- | --- |
| Процедура добавления элемента в дерево двоичного поиска | Рекурсивная процедура добавления элементов в дерево |
| procedure ADDDDP (root: tree; x: node);  var tek, pred: tree;  begin tek = root;  while tek <> nil do begin  pred := tek;  if x < tek^.info then tek := tek^.l else  tek := tek^.r; end;  new(tek); tek^.info := x; tek^.l := nil;  tek^.r := nil;  if x< pred^.info then pred^.l := tek else  pred^.r := tek;  end; | procedure Create (root: tree; x: node);  var tek: tree;  begin if x < root^.info then root^.l = nil  then begin new(tek);  tek^.info := x; tek^.l := nil;  tek^.r := nil; root^.l := tek;  end  else {}  Create (root^.l, x)  else if root^.r = nil then begin  new(tek);  tek^.info := x; tek^.l := nil; tek^.r := nil;  root^.r := tek; end  else Create (root^.r, x);  end; |

Построение дерева в основной программе:

var t: tree; x: node;

begin new(t); readln(x); t^.info := x;

t^.r := nil; t^.r := nil;

while x <> 0 do begin

readln(x);

if x <> 0 then create (t, x);

end;

***Операции с бинарными деревьями.***

Под обходом деревьев будем понимать последовательное посещение узлов с некоторой обработкой их информационной составляющей. Существуют три основных обхода дерева:

1. Прямой (П-обход). Состоит в посещении корня, П-обхода левого поддерева, П-обхода правого поддерева (корень→левое→правое).
2. Внутренний (В-обход). Состоит в посещении В-обходом левого поддерева, посещении корня, посещении В-обходом правого поддерева (левое→корень→правое).
3. Обратный (О-обход). Состоит в посещении О-обходом левого поддерева, посещении О-обходом правого поддерева, посещении корня (левое→правое→корень).

|  |  |  |
| --- | --- | --- |
| П-обход | В-обход | О-обход |
| Ob\_P (root: tree);  begin  if root <> nil then begin  writeln (root^.info);  Ob\_P(root^.l);  Ob\_P(root^.r);  end;  end; | Ob\_V (root: tree);  begin  if root <> nil then begin  Ob\_V(root^.l);  writeln (root^.info);  Ob\_V(root^.r);  end;  end; | Ob\_O (root: tree);  begin  if root <> nil then begin  Ob\_O(root^.l);  Ob\_O(root^.r);  writeln (root^.info);  end;  end; |

***Алгоритмы и задачи работы с множествами в языках программирования.***

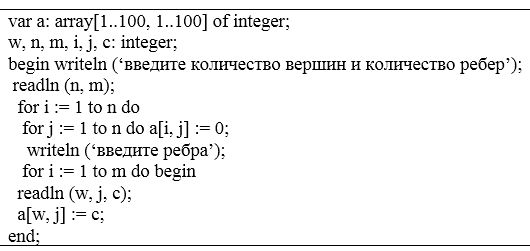
Любой множественный тип состоит из совокупности множеств, при этом элементы должны принадлежать некоторому базовому типу. В качестве базового типа используется любой скалярный или дискретный тип языка Паскаль. В большинстве реализаций базовый тип не должен содержать более 256 значений. Кроме того, можно использовать интервальные типы. Структура множественного типа имеет вид: set of <базовый тип>. Константные значения множественного типа отображаются значениями базового типа, заключенными в квадратные скобки: [‘B’, ‘+’, ‘D’]. В записи констант множеств можно использовать сокращения: [‘A’, ‘B’, ‘C’, ‘D’] → [‘A’..’D’]. Над множествами применимы следующие операции:

* Объединение (+);
* Пересечение (\*);
* Разность (\);
* Включение, принадлежность (in).

Множества можно сравнивать, при этом сравнение – это включение. В Паскале, как и в большинстве языках программирования, множества представляются логическими шкалами. Каждая переменная или константа множественного типа соответствует последовательности из нулей и единиц. Длина последовательности равна количеству элементов базового типа. Единицы стоят на тех местах последовательности, которые соответствуют элементам множеств. Объединение множеств – это поразрядная дизъюнкция логических шкал. Пересечение множеств – поразрядная конъюнкция. Для принадлежности необходимо построить шкалу элемента и выполнить поразрядную конъюнкцию. Если результат конъюнкции 0, то элемент не входит во множество, иначе – входит. На практике логические шкалы выбираются длиной, кратной байту.

***Программное представление графов.***

Ориентированный граф G = (V, E) состоит из множества узлов вершин V и множества ребер E. Ребра представляются парой (w, v), т.е. существует путь между w и v. Путем в ориентированном графе называется последовательность вершин , для которой существуют ребра . Вершина называется началом пути, вершина – концом пути. Длина пути – это количество ребер, составляющих путь (в данном случае n - 1). Путь называется простым, если все его вершины, кроме первой и последней, различны. Цикл – это простой путь длиной не менее единицы, начинающийся и заканчивающийся в одной вершине. На практике применяются помеченные графы, т.е. каждой дуге присваивается некий вес или стоимость, а каждая вершина помечена меткой. Одним из представлений графа является матрица смежности.

Определим множество вершин как V = {1…n}. Матрица смежности графа G – это матрица размером n на n, где элемент A[i, j] принимает значение true тогда и только тогда, когда существует дуга из i в j. Рассматривается помеченный граф. Элемент матрицы A[i, j] равен стоимости или весу дуги из i в j. На рисунке приведена реализация. Другим представлением ориентированного графа являются списки смежности. Списком смежности для вершины i является список всех вершин, смежных с i, причем определенным образом упорядоченный. В этом представлении существует массив Head, где элемент Head[i] является указателем на список смежности для вершины i. Матрица смежности обрабатывается за время O(n2), а список смежности за линейное время O(n).

***Реализация операций над графами.***

1. First(v). Возвращает индекс первой вершины, смежной с вершиной v. Если вершина v не имеет смежных вершин, возвращается нулевая вершина Ώ.

function First (v: integer): integer;

var i: integer; f: boolean;

begin f := true;

for i := 1 to n do

if a[v, i] and f then begin

first := i; f := false; end;

if f then write (‘вершина v не имеет смежных вершин’);

end;

1. Next(v, i). Возвращает индекс вершины, смежной с вершиной v, следующей за вершиной i. Если i – индекс последней вершины, смежной с вершиной, то возвращается Ώ.

function Next (v: integer; i: integer): integer;

var j, i: integer; f: boolean;

begin f := true;

for j := i + 1 to n do

if a[v, i] and f then begin

next := j; f := false; end;

if f then write (‘вершина v не имеет смежных вершин’);

end;

1. Vertex(v, i). Возвращает вершину с индексом i из множества вершин, смежных с v.

Последовательный просмотр вершин, смежных с вершиной v:

i := First(v);

while i <> 0 do begin

w := Vertex (v, i);

{действия с вершиной w}

i := Next (v, i);

end;

**Рекурсия.**

***Понятие рекурсии в языках программирования.***

Рекурсивной называется подпрограмма, которая в своем теле вызывает сама себя. Программирование простейших рекурсивных подпрограмм основано на понятии рекуррентных соотношений. Рекуррентное соотношение – некоторая формула, в которой вычисление значения для текущего аргумента зависит от значения при другом аргументе. Рекурсия может быть прямой или косвенной. При прямой рекурсии оператор вызова подпрограммы содержится непосредственно в ее исполняемой части. Любую рекурсивную процедуру (функцию) можно сделать не рекурсивной. Рекурсивное описание обычно короче и нагляднее, но требует больших затрат машинного времени (за счет повторных обращений) и памяти машины (за счет дублирования переменных).

***Рекурсивные процедуры и функции.***

У рекурсивной функции должны выполняться два свойства:

1. Должна присутствовать хотя бы одна не рекурсивная ветвь.
2. Внутри рекурсивной функции её вызов должен содержать формальный параметр по значению отличный от формального параметра заголовка.

В более сложных вариантах рекурсивная процедура входит в состав двух или более процедур, последняя из которых вновь обращается к начальной: А -> В -> А -> В -> А ... В язык Паскаль, требующий описать любой объект до его использования, пришлось включить специальное опережающее объявление рекурсивных процедур, сопровождаемое служебным словом forward:

{ Опережающее объявление первой в цепочке процедуры А }

procedure А(<список параметров>);

forward; { Описание последней в цепочке рекурсивной процедуры В }

procedure В(<список параметров>);

begin

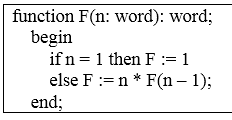
А(...); { Вызов рекурсивной процедуры А }

end; { Описание первой в цепочке рекурсивной процедуры А }

procedure A; { Здесь можно не повторять список аргументов } begin

В (...); { Вызов рекурсивной процедуры В } end;

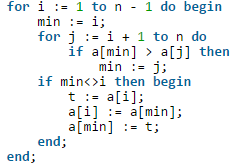
Несмотря на все изящество рекурсивных программ, их работа сопряжена с повышенными затратами машинного времени и ресурсов по памяти. При каждом новом вызове рекурсивной процедуры приходится сохранять значения всех ее локальных переменных и выделять новые участки памяти для очередной порции локальных данных. Как правило, рекуррентный алгоритм с большими или меньшими усилиями можно превратить в обычный циклический процесс. Однако, как бы ни относились разные программисты к рекурсии, следует признать, что некоторые рекурсивные программы выглядят очень эффектно и их алгоритмы оказываются намного более прозрачными.

***Механизм выполнения рекурсивных процедур.***

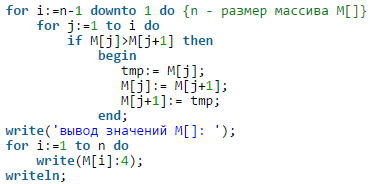
Рассмотрим рекурсивную функцию вычисления факториала числа n. Вычислим значение y := F(3). При вызове этой функции формируется модифицированное тело функции . Это модифицированное тело называется первой активацией функции. По ветке else порождается второй вызов функции, т.е. вторая активация, модифицированное тело которой имеет вид: . Проход по ветке else порождает третью активацию. В ней вычисление выполняется по ветке if, функция получает значение F := 1, активация завершается и уничтожается. Далее, вычисляется F(2), активация завершается и уничтожается, вычисляется F(3), активация завершается и уничтожается. Активации порождаются в порядке, обратном их завершению и уничтожению. Такой механизм называется стековым.

**Методы сортировки.**

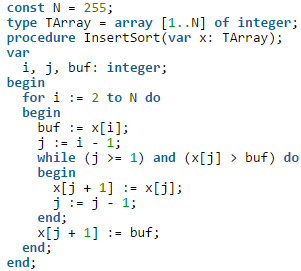
***Методы и алгоритмы сортировки одномерных массивов.***

1. Сортировка выбором.

Общая идея алгоритма состоит в том, что находится минимальный элемент массива данных и меняется местами с первым элементом массива. Т.о. на первом месте массива будет находится требуемый элемент. Далее алгоритм применяется к массиву начиная со второго элемента и т.д. Временная сложность алгоритма O(n2).

1. Сортировка обменом (алгоритм «пузырька»).

При первом проходе по массиву рассматривается первая запись и её ключ сравнивается поочередно с ключами других записей. Если встречается запись с большим ключом, то она меняется местами с первой и эта запись становится эталонной. Далее проверка выполняется с новым эталонным ключом. Временная сложность алгоритма O(n2).

1. Сортировка вставками (челночная сортировка).

На i-ом проходе по массиву элемент a[i] вставляют в необходимую позицию среди элементов a[1]…a[i-1], которые уже упорядочены. После этого первые i элементов будут упорядочены. Временная сложность алгоритма O(n2).

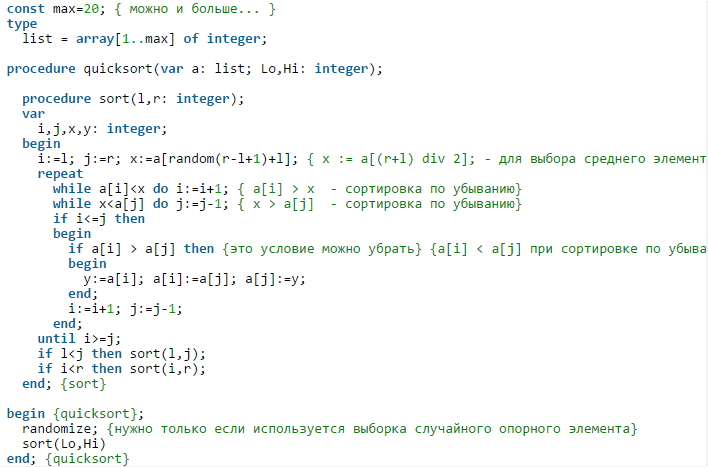
1. Сортировка подсчетом.

Особенность алгоритма состоит в том, что все элементы должны быть различны. В цикле подсчитывается количество элементов исходного массива, ключи которых меньше ключа первого элемента. В отсортированном массиве первый элемент должен занять позицию «найденное количество + 1». Временная сложность алгоритма O(n).

1. Метод парных сравнений.

На первом проходе сравниваются первый и второй элементы, затем третий и четвертый и т.д. Если необходимо, элементы в паре меняются местами. На втором проходе сравниваются второй и третий, четвертый и пятый элементы и т.д. Если необходимо, меняются местами. Проверки продолжаются до тех пор, пока ни при первом, ни при втором проходе не будет выполнено ни одного обмена. Временная сложность алгоритма O(n2).

1. Быстрая сортировка (сортировка Хоара).

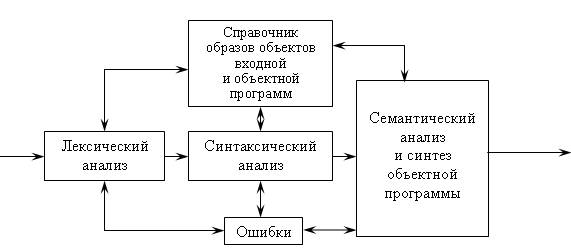
Можно предположить, что в массиве существует некоторый элемент f, такой, что все элементы, расположенные левее, имеют значение ключей меньше, а элементы, расположенные правее, имеют значение ключей больше. В этом случае можно говорить, что элемент f располагается на своем месте в отсортированном массиве. Элемент с номером f называется опорным, а его позиция – опорной позицией. Очевидно, когда опорный элемент найден, необходимо применить алгоритм независимо к левой и правой части массива. Опорный элемент получается путем перестановок. Одна из самых быстрых сортировок, позволяющая дать оценку времени выполнения O(n\*logn).

**Трансляторы языков высокого уровня.**

***Трансляторы языков программирования.***

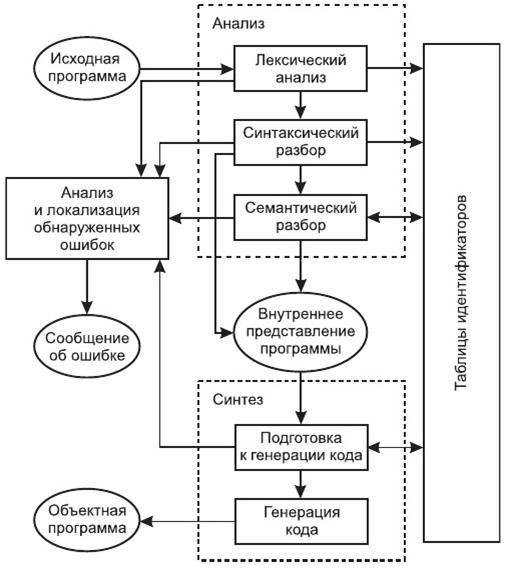
Любая программа, написанная на языке высокого уровня, должна быть сначала переведена в так называемый машинный эквивалент, т.е. программу, записанную на машинном языке и реализующую тот же алгоритм, что и исходная программа. Процесс такого перевода принято называть трансляцией. Очевидно, что в процессе трансляции каждая инструкция исходной программы (на языке высокого уровня) преобразуется в последовательность команд машинного эквивалента. Процесс трансляции может быть алгоритмизирован, т.е. можно написать программу, которая любую программу на языке высокого уровня переводит в машинный эквивалент. Такие программы-переводчики называют трансляторами. Существуют два основных типа трансляторов: компиляторы и интерпретаторы. В процессе компиляции строится машинный эквивалент исходной программы. Этот эквивалент запоминается в основной памяти либо во внешнем носителе и в последствии сформированный компилятор машинного эквивалента может быть выполнен столько раз, сколько необходимо при повторной компиляции. При интерпретации каждая инструкция программы переводится в машинный эквивалент. Этот эквивалент выполняется, после чего результат теряется. Затем указанные действия осуществляются с очередной инструкцией искомой программы.

***Назначение основных блоков транслятора, схема их взаимодействия и используемые ими структуры данных.***

 На рисунке показана упрощенная функциональная модель транслятора. В соответствии с этой моделью входная программа, прежде всего, подвергается лексической обработке. Цель лексического анализа – перевод исходной программы на внутренний язык компилятора, в котором ключевые слова, идентификаторы, метки и константы приведены к одному формату и заменены условными кодами: числовыми или символьными, которые называются дескрипторами. Каждый дескриптор состоит из двух частей: класса (типа) лексемы и указателя на адрес в памяти, где хранится информация о конкретной лексеме. Обычно эта информация организуется в виде таблиц. Одновременно с переводом исходной программы на внутренний язык на этапе лексического анализа проводится лексический контроль– выявление в программе недопустимых слов. Синтаксический анализатор воспринимает выход лексического анализатора и переводит последовательность образов лексем в форму промежуточной программы. Промежуточная программа является представлением синтаксического дерева программы. Последнее отражает структуру исходной программы, т.е. порядок и связи между ее операторами. В ходе построения синтаксического дерева выполняется синтаксический контроль– выявление синтаксических ошибок в программе. Фактическим выходом синтаксического анализа может быть последовательность команд, необходимых для того, чтобы строить промежуточную программу, обращаться к таблицам справочника, выдавать, когда это требуется, диагностическое сообщение. Синтез объектной программы начинается, как правило, с распределения и выделения памяти для основных программных объектов. Затем производится исследование каждого предложения исходной программы и генерируется семантически эквивалентные предложения объектного языка. В качестве входной информации здесь используется синтаксическое дерево программы и выходные таблицы лексического анализатора – таблица идентификаторов, таблица констант и другие. Анализ дерева позволяет выявить последовательность генерируемых команд объектной программы, а по таблице идентификаторов определяются типы команд, которые допустимы для значений операндов в генерируемых командах. Непосредственно генерации объектной программы часто предшествует семантический анализ, который включает различные виды семантической обработки. Один из видов – проверка семантических соглашений в программе, например, единственность описания каждого идентификатора в программе. Семантический анализ может выполняться на более поздних фазах трансляции, например, на фазе оптимизации программы, которая тоже может включаться в транслятор. Цель оптимизации – сокращение временных ресурсов или ресурсов оперативной памяти, требуемых для выполнения объектной программы

***Компиляторы.***

Компилятор – это программа, которая считывает текст программы, написанной на одном языке – исходном, и переводит его в эквивалентный текст на другом языке – целевом. В общем случае процесс компиляции можно представить схемой:

Процесс компиляции можно разбить на два основных этапа: анализ программы и синтез машинного эквивалента. На этапе анализа проверяется синтаксическая и семантическая корректность программы и выявляется её структура, т.е. определяются её отдельные компоненты: лексемы, разделители, операторы. Затем формируется промежуточная форма представления программы. На этапе синтеза выполняются подготовительные действия, в частности оптимизация промежуточной формы представления программы и генерация кода. Такой код называется объектным. Наиболее эффективным методом с точки зрения скорости компиляции является отображение исходной программы в абсолютную программу на машинном языке, пригодную для непосредственного исполнения. Такой тип компиляции больше всего подходит для небольших программ, не использующих независимо компилируемых подпрограмм. Однако для более сложных программ необходимо создавать объектные программы в форме последовательности перемещаемых машинных команд. Перемещаемая машинная команда представляет собой команду, в которой адресация ячеек памяти производится относительно некоторого подвижного начала. Объектную программу называют также перемещаемым объектным сегментом. Этот сегмент может быть связан с другими сегментами, такими, как независимо компилируемые подпрограммы пользователя, подпрограммы ввода-вывода и библиотечные функции. Такое связывание, т.е. создание единого перемещаемого объектного сегмента из набора различных сегментов, осуществляется программой, которая называется редактором связей. Далее единый перемещаемый объектный сегмент или модуль загрузки размещается в памяти программой, называемой загрузчиком, которая превращает перемещаемые адреса в абсолютные. После этого программа готова к выполнению. Приведенная схема связывания называется статическим связыванием. В настоящее время часто используется также динамическое связывание. Это означает, что объектный сегмент не включается в единый перемещаемый объектный сегмент до тех пор, пока не произойдет обращение к этому сегменту. Выбор статического или динамического связывания зависит от операционной системы. Динамическое связывание часто используется для того, чтобы разрешить нескольким выполняемым программам разделять одну копию подпрограммы.

***Основные функции компилятора.***

Компилятор с точки зрения теории формальных языков выполняет две основные функции:

1. является распознавателем для языка исходной программы. Получает на вход цепочку символов входного языка, проверяет ее принадлежность языку и выявляет правила, по которым эта цепочка построена;
2. генерирует результирующую программу. На выходе создается цепочка выходного языка по определенным правилам. Распознавателем сгенерированной цепочки объектной программы будет выступать вычислительная система.

Кратко представим функции основных фаз компиляции:

Лексический анализ. Эту часть компилятора выполняет сканер, который читает литеры программы (символы) на исходном языке и строит из них слова (лексемы) исходного языка. На входе сканера (лексического анализатора) текст исходной программы, выходная информация передается для дальнейшей обработки на этап синтаксического разбора.

Синтаксический разбор − это основная часть компилятора на этапе анализа. Здесь в тексте исходной программы выделяются синтаксические конструкции. Кроме того, проверяется синтаксическая правильность программы.

Семантический анализ − это часть компилятора, проверяющая часть текста исходной программы с точки зрения семантики входного языка.

Подготовка к генерации кода − на этой фазе компилятор выполняет предварительные действия, непосредственно связанные с синтезом текста результирующей программы: идентификация элементов языка, распределение памяти и т.п. Эта подготовка ещё не ведёт к порождению текста на выходном языке.

Генерация кода − это фаза, на которой непосредственно порождаются команды, составляющие предложения выходного языка и текст результирующей программы в целом. Фаза генерации кода основная на этапе синтеза результирующей программы. Кроме этого, генерация обычно включает в себя и оптимизацию. Оптимизация − это процесс, связанный с обработкой уже порожденного текста и оказывающий существенное влияние на качество и эффективность результирующей программы.

**Формальные средства описания языков программирования.**

***Описание языков программирования с помощью БНФ.***

БНФ состоит из следующих обозначений:

< > – в такие скобки заключаются понятия, которые требуют определения.

::= – читается как «есть» и разделяет левую и правую часть определяемого понятия.

│ – читается как «или» и служит для разделения альтернатив.

[ ] – заключаемые понятия могут присутствовать либо отсутствовать в данной конструкции.

{ } – заключаемые понятия могут отсутствовать либо повторяться любое конечное число раз.

<двоичная цифра>::=0 | 1.

<двоичный код>::=<двоичная цифра> |<двоичный код><двоичная цифра>

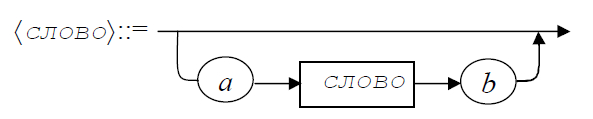
***Описание языков программирования с помощью синтаксических диаграмм.***

Язык синтаксических диаграмм – это графическая форма представления описания языка, включающая элементы:

 – записывается определенная информация.

 – содержат понятия, не требующие определения.

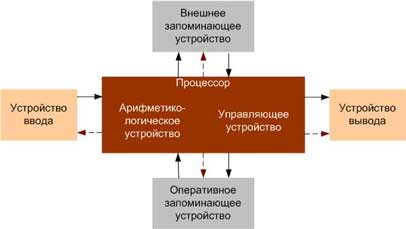
 – указывают последовательность прямоугольников и овалов.

Понятие слова, состоящего из цепочек букв a и b.

**Основы архитектуры процессоров Intel.**

***Основы архитектуры ЭВМ.***

Архитектура ЭВМ – это наиболее общие принципы построения ЭВМ, реализующие программное управление работой и взаимодействием основных ее функциональных узлов. Джон Фон Нейман выдвинул основополагающие принципы логического устройства ЭВМ и предложил ее структуру:  
Арифметико-логическое устройство и устройство управления в современных компьютерах образуют процессор ЭВМ. Процессор, который состоит из одной или нескольких больших интегральных схем называется микропроцессором или микропроцессорным комплектом.

Процессор – функциональная часть ЭВМ, выполняющая основные операции по обработке данных и управлению работой других блоков. Процессор является преобразователем информации, поступающей из памяти и внешних устройств.

Запоминающие устройства обеспечивают хранение исходных и промежуточных данных, результатов вычислений, а также программ. Они включают: оперативные (ОЗУ), сверхоперативные СОЗУ), постоянные (ПЗУ) и внешние (ВЗУ) запоминающие устройства.

Оперативные ЗУ хранят информацию, с которой компьютер работает непосредственно в данное время (резидентная часть операционной системы, прикладная программа, обрабатываемые данные). В СОЗУ хранится наиболее часто используемые процессором данные. Только та информация, которая хранится в СОЗУ и ОЗУ, непосредственно доступна процессору.

Внешние запоминающие устройства (накопители на магнитных дисках, например, жесткий диск или винчестер) с емкостью намного больше, чем ОЗУ, но с существенно более медленным доступом, используются для длительного хранения больших объемов информации. Например, операционная система (ОС) хранится на жестком диске, но при запуске компьютера резидентная часть ОС загружается в ОЗУ и находится там до завершения сеанса работы ПК.

ПЗУ (постоянные запоминающие устройства) и ППЗУ (перепрограммируемые постоянные запоминающие устройства) предназначены для постоянного хранения информации, которая записывается туда при ее изготовлении, например, ППЗУ для BIOS.

В качестве устройства ввода информации служит, например, клавиатура. В качестве устройства вывода – дисплей, принтер и т.д.

***Номенклатура сегментных регистров.***

В программной модели микропроцессора имеется шесть сегментных регистров: cs, ss, ds, es, gs, fs. Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых сегментами. Соответственно такая организация памяти называется сегментной. Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры. Фактически этих регистрах содержатся адреса памяти, с которых начинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку, неявно используются адреса во вполне определенных сегментных регистрах. Микропроцессор поддерживает следующие типы сегментов:

1. Сегмент кода. Содержит команды программы. Для доступа к этому сегменту служит регистр cs (code segment) – сегментный регистр кода. Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (т. е. эти команды загружаются в конвейер микропроцессора).
2. Сегмент данных. Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр ds (data segment) – сегментный регистр данных, который хранит адрес сегмента данных текущей программы.
3. Сегмент стека. Этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этому сегменту служит регистр ss (stack segment) – сегментный регистр стека, содержащий адрес сегмента стека.
4. Дополнительный сегмент данных. Не явно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре ds, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в регистрах es, gs, fs (extension data segment).

***Варианты задания исполнительного адреса в архитектурах Intel или др.***

Минимальной адресуемой единицей основной памяти является байт, состоящий из 8 бит. Адресовать отдельно бит нельзя. Если необходимо получить доступ к определенному биту, то сначала ищется соответствующий байт, а затем уже в нем – нужный бит. Номер байта является его физическим адресом в устройстве памяти. Для размещения программ и данных в основной памяти выделяются специальные области – сегменты. Сегмент при 16-ти разрядной адресации – фрагмент памяти, начинающийся с адреса кратного 16 и имеющий размер от 1 байта до 64 Кб. Следовательно, базовый адрес сегмента всегда содержит в 4-х младших разрядах нули. Старшая часть адреса сегмента без последних четырех нулей называется сегментным адресом и хранится в одном из 4-х сегментных регистров. Физический адрес любых данных в памяти формируется из 16-ти битового смещения и 16-ти битового сегментного адреса по специальной схеме. Сначала к сегментному адресу аппаратно дописываются 4 двоичных нуля. В результате получается 20-ти битовый физический адрес начала сегмента. Затем выполняется сложение 20-ти битового базового адреса сегмента и 16-ти битового смещения. Т.о., для адресации основной памяти предусматриваются 20-ти битовые адреса, что позволяет работать с основной памятью до 1 Мб. Если программа включает более чем в один сегмент кодов, данных и стека, то сегментные регистры в процессе ее работы перегружаются. Смещение для каждого типа сегмента формируется по своим правилам. Для стека смещение хранится в регистре SP, для сегмента кодов – в IP, а для сегментов данных – рассчитывается в соответствие с форматом команды, как исполнительный адрес. Стек при 16-ти разрядной адресации представляет собой специальным образом организованную область памяти, осуществляющую последовательную запись элементов данных длиной 2 байта (слово) и чтение их в порядке, обратном порядку записи. Для хранения адреса последнего слова, занесенного в стек, служит регистр-указатель стека SР. Каждый раз при записи данных значение SP уменьшается на 2, а при чтении – увеличивается на 2. Т.о., стек растет в область младших адресов, и в начальный момент времени указатель SP должен содержать максимально возможное для конкретного размера стека смещение. Стек используется для временного хранения данных и адресов, например, при вызове подпрограмм, когда в стек заносится адрес возврата и значения параметров, передаваемых в подпрограмму. Принципиально допускается 8 способов задания исполнительного адреса операндов, размещенных в основной памяти:

1. SI + <смещение, заданное в команде>;
2. DI + <смещение, заданное в команде>;
3. BP + <смещение, заданное в команде>;
4. BХ + <смещение, заданное в команде>;
5. BP + SI + <смещение, заданное в команде> ;
6. BP + DI + <смещение, заданное в команде> ;
7. BX + SI + <смещение, заданное в команде>;
8. BX + DI + <смещение, заданное в команде>.

Во всех случаях исполнительный адрес операнда определяется как сумма содержимого указанных регистров и смещения, заданного в команде и представляющего собой одно или двухбайтовое число.

**Языки программирования ассемблерного типа.**

***Особенности языков ассемблерного типа.***

Основные особенности ассемблера:

* вместо двоичных кодов в языке используются символьные имена – мнемо­ника. Например, для команды сложения (add) используется мнемоника ADD, вычитания (subtract) – SUB, умножения (multiply) – MUL, деления (divide) – DIV и т. д. Символьные имена используются и для адресации ячеек памяти. Для программирования на языке ассемблера вместо двоичных кодов и адре­сов нужно знать только символические названия, которые ассемблер транс­лирует в двоичные коды;
* каждое высказывание соответствует одной машинной команде (коду), т. е. су­ществует взаимно однозначное соответствие между машинными командами и операторами в программе на языке ассемблера;
* язык обеспечивает доступ ко всем объектам и командам. Языки высокого уровня такой способностью не обладают. Например, язык ассемблера позво­ляет выполнить проверку бита регистра флагов, а язык высокого уровня такой способностью не обладает. Языки для сис­темного программирования часто занимают промежуточное положение. С точки зрения возможностей доступа они ближе к языку ассемб­лера, однако обладают синтаксисом языка высокого уровня;
* язык ассемблера не является универсальным языком. Для каждой определен­ной группы микропроцессоров имеется свой ассемблер. Языки высокого уровня лишены этого недостатка.

В отличие от языков высокого уровня написание и отладка программы на язы­ке ассемблера занимает много времени. Несмотря на это, язык ассемблера по­лучил широкое распространение в силу следующих обстоятельств:

* составленная на языке ассемблера программа имеет значительно меньшие размеры и работает гораздо быстрее, чем программа, написанная на языке высокого уровня. Для некоторых приложений эти показатели играют перво­степенную роль, например, многие системные программы, программы в кредитных карточках, сотовых телефонах, драйверах устройств и др.;
* некоторым процедурам требуется полный доступ к аппаратному обеспече­нию, что обычно невозможно осуществить на языке высокого уровня. К этому случаю относятся прерывания и обработчики прерываний в операционных системах, а также контроллеры устройств во встроенных системах, работа­ющих в режиме реального времени.

***Макросредства языка Ассемблер (на примере архитектуры Intel или др.).***

В схеме использования подпрограмм, независимо от количества их вызова, в основной памяти код подпрограммы находится в единственном экземпляре. В общем случае, этот код может находиться на существенном удалении от программы. В тех случаях, когда код подпрограммы небольшой, его целесообразно подставить в точку вызова программы. Подпрограммы, код которых присутствует в единственном экземпляре в памяти независимо от количества вызовов, называются закрытыми или замкнутыми. Подпрограммы, код которых подставляется в точку вызова, называются открытыми или макросами. Текст открытой подпрограммы называется макроопределением. Её вызов называется макрокомандой. Макроопределения могут быть параметризированы, при этом параметры определяются во время компиляции. Процесс замены макрокоманды на соответствующее макроопределение с возможной заменой параметров на фактические значения называют процессом макрорасширения. Макрорасширением также называют текст, получающийся в процессе замены макрокоманды соответствующим макроопределением. Процесс макрорасширения выполняется до компиляции и представляет собой некоторое преобразование текста программы. Программы, преобразующие текст по заданным правилам, называются текстовыми процессорами. Т.к. макрорасширение выполняется до компиляции, то такой текстовый процессор называется препроцессором. Поскольку этот препроцессор обрабатывает макрокоманды и макроопределения, его называют макропроцессором. Совокупность макропроцессора и ассемблера называется макроассемблером (MASM).

**Операционные системы.**

***Операционные системы, их назначение и функции.***

Операционная система – это программа, которая обеспечивает возможность рационального использования оборудования компьютера удобным для пользователя образом. Это базовый комплекс компьютерных программ, обеспечивающий управление аппаратными средствами компьютера, работу с файлами, ввод и вывод данных, а также выполнение прикладных программ и утилит. Выделяют ряд различных «точек зрения» на операционные системы:

1. ОС как виртуальная машина. Архитектура ПК на уровне машинных команд неудобна для использования прикладными программами. Поэтому существует некая высокоуровневая абстракция, с помощью которой от программиста скрывают подробности обработки сообщения и прерываний, и операционная система выступает как некая виртуальная машина.
2. ОС как система управления ресурсами. Операционная система позволяет обращаться к ресурсам вычислительной системы и основной функцией операционной системы является синхронизация обращения задач к ресурсам.
3. ОС как защитник пользователей и программ. Вычислительная система допускает одновременную работу нескольких пользователей, поэтому операционная система должна обеспечить безопасную совместную работу.
4. ОС как постоянно функционирующее ядро. Во всех операционных системах выделяется некоторая часть, которая постоянно работает на компьютере и называется ядром операционной системы.

***Виды операционных систем, их характеристики.***

Операционные системы классифицируются по различным признакам:

1. Поддержка многозадачности.

По числу одновременно выполняемых задач выделяют однозадачные (MS DOS) и многозадачные.

1. Поддержка многопользовательского режима.

По числу одновременно работающих пользователей выделяют однопользовательские (Windows 3x) и многопользовательские.

1. Вытесняющая и не вытесняющая многозадачность.

Способ распределения процессорного времени между задачами определяет две группы алгоритмов: невытесняющая и вытесняющая многозадачность. В первом случае механизм планирования процессов целиком определен операционной системой. Во втором случае механизм планирования распределен между операционной системой и прикладными программами. В первом случае процесс активен до тех пор, пока сам не передаст управление операционной системе. (Второй лучше).

1. Системы реального времени.

Это свойство определяет возможность распараллеливания задач. В этом случае операционная система распределяет процессорное время между подзадачами.

1. Многопроцессорная обработка.

Наличие этого свойства приводит к усложнению управления операционной системой. Многопроцессорные операционные системы подразделяются на симметричные и ассиметричные системы. В ассиметричных один процессор управляется операционной системой, остальные – прикладными программами. Симметричная система децентрализована.

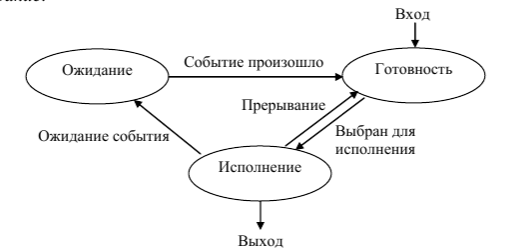
***Управление процессами.***

Важнейшей частью ОС является подсистема управления процессами. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

***Понятие процесса, состояния процесса.***

Процесс – это программа пользователя в ходе ее выполнения в компьютерной системе. Процесс представляется некоторой структурой данных, содержащих следующую информацию:

1. Состояние, в котором находится процесс.
2. Программный счетчик процесса, т.е. адрес команды, которая должна быть выполнена в следующий момент.
3. Содержимое регистров процесса.
4. Данные, необходимые для работы процесса (уровень приоритета процесса, размер и местоположение адресного пространства).
5. Учетные данные (идентификационный номер процесса, данные о пользователе процесса и т.д.).
6. Сведения об устройства ввода/вывода (таблица открытых файлов; список устройств, закрепленных за процессом).

Рассмотрим схему работы процесса:

Всякий новый процесс, появляющийся в системе, попадает в состояние готовности. Операционная система в соответствии с алгоритмами планирования выбирает один из готовых процессов и переводит его в состояние исполнения. Из состояния исполнения процесс может выйти по трем причинам:

1. Операционная система прекращает деятельность процесса.
2. Процесс не может продолжать исполнение пока не произойдет некоторое событие, тогда операционная система переводит его в состояние ожидания.
3. В результате возникшего прерывания, например, от таймера. Процесс переходит в состояние ожидания.

Из состояния ожидания процесс переходит в состояние готовности после того, как ожидаемое событие произошло.

***Операции над процессами.***

Переходом процесса из одного состояния в другое управляет операционная система.

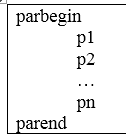
Операции над процессами удобно разбить на пары:

1. Создание – завершение процесса.
2. Приостановка – запуск процесса.
3. Блокирование – разблокирование процесса.

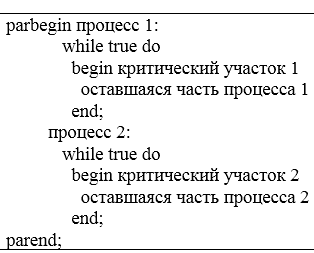
Операции создания и завершения процесса являются одноразовыми. Все остальные операции могут выполняться неоднократно и называются многоразовыми. Приостановка процесса происходит в результате прерывания, блокировка процесса – это ожидание некоторого события.

***Асинхронные параллельные процессы.***

Для процессов, выполняющих одновременно некоторые действия, будем использовать следующую конструкцию:

Процессы выполняются независимо друг от друга и, возможно, с разными скоростями. Работа процессов, использующих общие критические ресурсы, должна быть синхронизирована. Проблема критических участков будет разрешима, если выполняются условия:

1. Если несколько процессов пытаются войти в свой критический участок, то это допустимо только для одного процесса.
2. Если несколько процессов пытаются войти в свой критический участок, то за конечное время это право будет предоставлено хотя бы одному процессу.

Два процесса, содержащие критический участок, взаимодействуют по схеме.

***Взаимоисключение.***

Способ взаимодействия, при котором во время обращения одного процесса к разделяемым данным всем другим процессам это запрещается, называется взаимоисключением.

1. Программные алгоритмы организации взаимодействия процессов.

Решение этой группы алгоритмов основано на использовании переменных и , каждая из которых может принимать значение 0 или 1 и с помощью которых процесс сигнализирует другому процессу о своём намерении войти в критический участок. Однако эта схема не работоспособна, если скорость выполнения первого процесса намного выше скорости второго.

1. Алгоритм Деккера.

В этом алгоритме кроме переменных и существует специальная управляющая переменная очередь, которая указывает, какой из двух процессов должен войти в свой критический участок.

1. Операции проверка и установка.

В аппаратной части компьютера существует команда, семантика которой описывается схемой: ПР\_УСТ(L, C): L := C; C := 1;. Основное свойство операции – её неделимость. L – локальная переменная процесса, а C – глобальная по отношению к процессам. В задаче синхронизации критического участка проблема синхронизации решается за счет необходимости активного ожидания каждого из процессов.

1. Cемафоры Дейкстры.

Семафор S – переменная с целыми неотрицательными значениями, доступ к которой любого процесса осуществляется с помощью операции P(s) и V(s). Эти операции являются неделимыми (атомарными).

В момент создания семафор инициализируется любым положительным значением. Операции P(s) и V(s) могут быть реализованы как обработчики прерываний, в процессе выполнения которых другие прерывания запрещены. Очередь может быть реализована любой информационно-динамической структурой, позволяющей помещать и извлекать процесс в очередь. Семафор с максимальным значением 1 называется двоичным.

**Технологии программирования.**

***Объектно-ориентированное программирование.***

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определенного [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а классы образуют иерархию наследования.

Кроме этого, объектный подход предлагает новые технологические средства разработки, такие как наследование, полиморфизм, композиция, наполнение, позволяющие конструировать сложные объекты из более простых. В результате существенно увеличивается показатель повторного использования кодов, появляется возможность создания библиотек объектов для различных применений, и разработчикам предоставляются дополнительные возможности создания систем повышенной сложности. Основной недостаток ООП – некоторое снижение быстродействия за счет более сложной организации программной системы.

***Классы и объекты.***

В ООП определены следующие ключевые свойства объектов.

• Поведение (behavior) объекта— что с ним можно делать и какие методы к нему можно применять.

• Состояние объекта — как этот объект реагирует на применение методов.

•Сущность (identity) объекта — чем данный объект отличается от других, характеризующихся таким же поведением и состоянием.

Все объекты, являющиеся экземплярами одного и того же класса, ведут себя одинаково. Поведение объекта определяется методами, которые можно вызвать. Далее, каждый объект сохраняет информацию о своем состоянии. Со временем состояние объекта может измениться, однако спонтанно это произойти не может. Состояние объекта может изменяться только в результате вызовов методов. (Если состояние объекта изменилось вследствие иных причин, значит, принцип инкапсуляции не соблюден.)

Состояние объекта не полностью описывает его, поскольку каждый объект имеет свою собственную сущность.

Класс  – это множество объектов, связанных общностью свойств, поведения, связей и семантики. Любой объект является экземпляром класса. Это шаблон. Объекты с одинаковыми свойствами, то есть с одинаковыми наборами переменных состояния и методов, образуют класс. Конструктор класса  – специальный блок инструкций, вызываемый при создании объекта. Деструктор  –  специальный метод класса, служащий для деинициализации объекта (например, освобождения памяти). Атрибут — поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства. Атрибуты могут быть скрыты от других классов, это определяет видимость атрибута: рublic (общий, открытый); private (закрытый, секретный); protected (защищенный).

***Наследование.***

Наследование – концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения. В [объектно-ориентированном программировании](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) абстрактные типы данных называются [классами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Класс, определённый через наследование от другого класса, называется: производным классом, классом потомком или подклассом. Класс, от которого новый класс наследуется, называется: предком, базовым классом или суперклассом. Наследование является механизмом повторного использования кода и способствует независимому расширению программного обеспечения через открытые классы и интерфейсы. Установка отношения наследования между классами порождает иерархию классов. Виды наследования:

Простое. При простом наследовании у класса не более одного класса-предка.

Множественное. При множественном наследовании, у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Недостаток в возникновении ошибок из-за возможного наличия одинаковых имён методов в предках. Большинство современных объектно-ориентированных языков программирования поддерживают возможность одновременно наследоваться от класса-предка и реализовать методы нескольких интерфейсов одним и тем же классом. Этот механизм позволяет во многом заменить множественное наследование – методы интерфейсов необходимо переопределять явно, что исключает ошибки при наследовании функциональности одинаковых методов различных классов-предков.

***Полиморфизм.***

Полиморфизм – взаимозаменяемость объектов с одинаковым интерфейсом. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию – например, реализация класса может быть изменена в процессе наследования.

Возможность приписывать разную функциональность одному методу (функции, операции) называется [перегрузкой](http://www.sbup.com/w/index.php?title=%D0%9F%D0%B5%D1%80%D0%B5%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B9&action=edit&redlink=1) метода (функций, [операций](http://www.sbup.com/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2)).

Полиморфизм реализуется с помощью наследования классов и виртуальных функций. Класс-потомок наследует сигнатуры методов класса-родителя, но реализация этих методов может быть другой, соответствующей специфике класса-потомка. Это называется переопределением метода. Другие функции могут работать с объектом класса-родителя, при этом вместо него во время исполнения будет подставляться один из классов-потомков. Это называется поздним связыванием. Класс-потомок сам может быть родителем. Это позволяет строить сложные схемы наследования – древовидные или сетевидные. Абстрактные (или чисто виртуальные) методы не имеют реализации вообще (на самом деле некоторые языки, допускают реализацию абстрактных методов в родительском классе). Они специально предназначены для наследования. Их реализация должна быть определена в классах-потомках.

***Технологии программирования.***

Технологией программирования называют совокупность методови средств, используемых в процессе разработки программного обеспечения. Технология программирования представляет собой набор технологических инструкций, включающих:

− указание последовательности выполнения технологическихопераций;

− перечисление условий, при которых выполняется та или инаяоперация;

− описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы,стандарты, критерии и методы оценки и т.п.

Кроме набора операций и их последовательности, технологиятакже определяет способ описания проектируемой системы, точнеемодели, используемой на конкретном этапе разработки.Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии,охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый метод, позволяющий решить конкретную задачу. В основе вторых обычно лежит базовый метод или подход, определяющий совокупность методов,используемых на разных этапах разработки, или методологию.

***Основные технологические подходы.***

1. Ранние технологические подходы. Не используют явных технологий. Обычно их применяют только для очень маленьких проектов, завершающихся созданием демонстрационного прототипа.
2. Подход "кодирование-исправление". Разработчик начинает кодирование системы с самого первого дня, не занимаясь серьезным проектированием. Все ошибки обнаруживаются к концу кодирования и требуют исправления через повторное кодирование. При использовании данного подхода затрачивается время лишь на кодирование и заказчику легко демонстрировать прогресс в разработке в строках кода.
3. Каскадные технологические подходы. Задают некоторую последовательность выполнения процессов, изображаемую в виде каскада. Их ещё называют подходами на основе модели водопада.
4. Каскадный подход. Специфика "чистого" каскадного подхода такова, что переход к следующему процессу осуществляется только после того, как завершена работа с текущим процессом. Возвраты к уже пройденным процессам не предусмотрены. Данный подход рекомендован к применению в тех проектах, где в самом начале все требования могут быть сформулированы точно и полно.
5. Каскадно-возвратный подход. Разрешены возвраты к предыдущим стадиям и пересмотр или уточнение ранее принятых решений. Каскадно-возвратный подход отражает итерационный характер разработки программного обеспечения. Этот подход в значительной степени отражает реальный процесс создания программного обеспечения, в том числе и существенное запаздывание с достижением результата. На задержку оказывают влияние корректировки при возвратах.
6. Каскадно-итерационный подход. Предусматривает последовательные итерации каждого процесса до тех пор, пока не будет достигнут желанный результат. Каждая итерация является завершенным этапом, и ее итогом будет некоторый конкретный результат.
7. Каскадный подход с перекрывающимися процессами. Предполагает наличие специализированных команд, позволяющих до определенной степени сократить передаваемую документацию. Следующий процесс начинается до завершения текущего. Несколько процессов могут выполняться параллельно.
8. Каскадный подход с подпроцессами. Особенность его в том, что с архитектурной точки зрения проект достаточно часто может быть разделен на подпроекты, которые могут разрабатываться индивидуально. В данном подходе требуется дополнительная фаза тестирования подсистем до объединения их в единую систему.
9. Спиральная модель. Использует понятие прототипа – программы, реализующей частичную функциональность создаваемого программного продукта. Создание прототипов осуществляется за несколько витков спирали, каждый из которых состоит из "анализа риска", "некоторого процесса" и "верификации". Обращение к каждому процессу предваряет "анализ риска", причем, если риск превышения сроков и стоимости проекта оказывается существенным, то разработка заканчивается.

***Жизненный цикл программного продукта.***

Жизненный цикл программного продукта – период существования программного продукта, исчисляемый от начала его проектирования до уничтожения. Основные этапы жизненного цикла:

1. Маркетинг рынка программных изделий, спецификация требования к программному изделию;
2. Проектирование структуры программного продукта;
3. Программирование, тестирование, автономная и комплексная отладка программ;
4. Подготовка программной документации;
5. Выход на рынок программных изделий, распространение программного продукта;
6. Эксплуатация программного продукта пользователями;
7. Сопровождение программного продукта;
8. Снятие программного продукта с продажи, отказ от сопровождения.

Стандарт описывает 17 процессов жизненного цикла, распределенных по трем категориям:

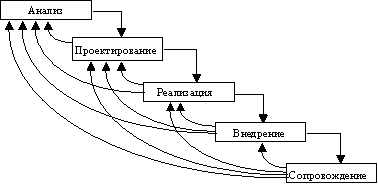
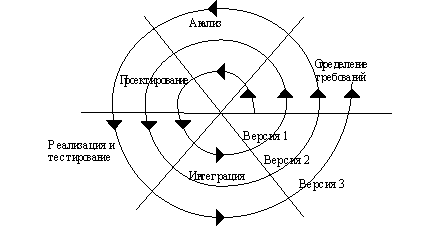
* Основные:
  + Приобретение (действия и задачи заказчика, приобретающего ПО);
  + Поставка (действия и задачи поставщика, который снабжает заказчика программным продуктом или услугой);
  + Разработка (действия и задачи, выполняемые разработчиком);
  + Эксплуатация (действия и задачи оператора – организации, эксплуатирующей систему);
  + Сопровождение (действия и задачи, выполняемые сопровождающей организацией). Сопровождение – внесений изменений в ПП в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.
* Вспомогательные:
* Организационные:

***Модели ЖЦ ПП.***

Наибольшее распространение получили следующие две основные модели ЖЦ: каскадная (70-85 г.г.) и спиральная модель (86-90 г.г.). Основной характеристикой каскадного способа является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков. Положительные стороны применения каскадного подхода заключаются в следующем:

* на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
* выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, чтобы предоставить разработчикам реализовать их как можно лучше с технической точки зрения. Однако, в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПП никогда полностью не укладывался в такую жесткую схему, т.к. постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПП принимал следующий вид:

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС "заморожены" в виде технического задания на все время ее создания. Т.о., пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПП, пользователи получают систему, не удовлетворяющую их потребностям. Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ, делающая упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПП, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации. Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная задача быстрее показать пользователям системы работоспособный продукт, активизируя процесс уточнения и дополнения требований. Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

**Компьютерные сети.**

***Компьютерные сети.***

Компьютерная сеть (Computer Network) – это множество компьютеров, соединенных линиями связи и работающих под управлением специального программного обеспечения. Под линией связи обычно понимают совокупность технических устройств, и физической среды, обеспечивающих передачу сигналов от передатчика к приемнику. В реальной жизни примерами линий связи могут служить участки кабеля и усилители, обеспечивающие передачу сигналов между коммутаторами телефонной сети. На основе линий связи строятся каналы связи. Каналом связи обычно называют систему технических устройств и линий связи, обеспечивающую передачу информации между абонентами. Соотношение между понятиями "канал" и "линия" описывается следующим образом: канал связи может включать в себя несколько разнородных линий связи, а одна линия связи может использоваться несколькими каналами. Главной целью объединения компьютеров в сеть является предоставление пользователям возможности доступа к различным информационным ресурсам, распределенным по этим компьютерам и их совместного использования. Важной характеристикой любой компьютерной сети является широта территории, которую она охватывает. Широта охвата определяется взаимной удаленностью компьютеров, составляющих сеть и влияет на технологические решения, выбираемые при построении сети.

***Типы каналов связи.***

Канал – физическая среда для передачи сигналов, передающих информацию; включает оборудование; программное обеспечение. Каналы делятся на проводные (сигнал передается по проводнику) и беспроводные (физической средой является радиоэфир, и информация передается электрическими волнами). Чем короче волна проводного канала, тем уязвимее сигнал. По направлению передачи информации каналы делятся на:

1. Симплексные. Позволяют передавать информацию только в одном направлении.
2. Полудуплексные. Позволяют передавать информацию попеременно в прямом и обратном направлении.
3. Дуплексные. Одновременно можно передавать информацию в разных направлениях.

По способу использования каналы делятся на:

1. Выделенные (арендованные). Закреплены за двумя абонентами на определенный промежуток времени.
2. Коммутируемые. Закрепляются за абонентами только на время сеанса связи.

По виду передаваемых сигналов каналы делятся на.

1. Аналоговые. Данные передаются в виде синусоидальных гармонических колебаний. Передача информации осуществляется за счет методов модуляции. Кодирование данных проводят, используя амплитудную, частотную и фазовую модуляцию.
2. Цифровые. Передача информации осуществляется в импульсном виде. При таком способе нет необходимости преобразования сигналов в аналоговые и обратно. Используют разные способы кодирования. Методы кодирования должны отвечать требованиям простоты, самосинхронизации, использования одного уровня напряжения, максимального использования полосы пропускания данных.

***Эталонная модель OSI.***

Open System Interconnection разработана международной организацией по стандартам ISO.

Данные, передаваемые от источника к адресату, проходят семь уровней. На каждом уровне выполняется определенная задача, что в итоге гарантирует доставку данных в конечный пункт и делает их передачу независимой от применяемых средств. Таким образом, достигается совместимость между сетями с разными топологиями и сетевым оборудованием.

**Физический уровень** – участвует в организации связи, учитывая особенности среды передачи данных. Кроме того, отвечает за передачу потока информации и преобразование ее в соответствии с существующими методами кодирования.

**Канальный уровень** обеспечивает гарантированную передачу данных с помощью алгоритмов физического уровня и проверяет корректность полученных данных. Может реализовываться как аппаратно (например, с помощью коммутаторов), так и с помощью программного обеспечения (например, драйвера сетевого адаптера).

**Сетевой уровень** необходим для определения кратчайшего пути и преобразования логических адресов и имен сетевых устройств в их физическое представление (процесс маршрутизации).

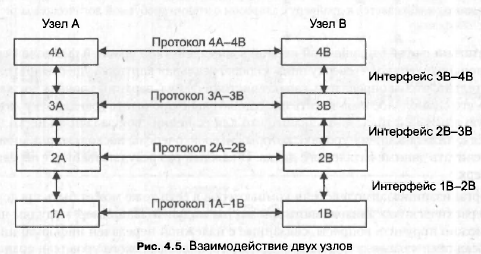
**Транспортный уровень** используется для организации надежной передачи данных, которая исключает потерю информации, ее некорректность или дублирование.

**Сеансовый уровень** отвечает за создание, сопровождение и поддержание сеанса связи на время, необходимое для завершения передачи всего объема данных и производит синхронизацию передачи пакетов, осуществляя проверку доставки и целостности пакета.

**Уровень представления данных** (представительский уровень) преобразует данные из формата для передачи по сети в формат, понятный более высокому уровню, и наоборот. Именно на этом уровне применяются алгоритмы шифрования и сжатия данных.

**Прикладной уровень** отвечает за связь сети с пользователями – приложениями, которым требуется информация от сетевых служб всех уровней. С его помощью можно узнать все, что происходило в процессе передачи данных, а также информацию об ошибках, возникших в процессе их передачи.

***Понятия: протокол и интерфейс.***

На рисунке 4.5 показана модель взаимодействия двух узлов. С каждой стороны средства взаимодействия представлены четырьмя уровнями. Процедура взаимодействия этих двух узлов может быть описана в виде набора правил взаимодействия каждой пары соответствующих уровней обеих участвующих сторон. Формализованные правила, определяющие последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне, но в разных узлах, называются протоколом. Модули, реализующие протоколы соседних уровней и находящиеся в одном узле, также взаимодействуют друг с другом в соответствии с четко определенными правилами с помощью стандартизированных форматов сообщений. Эти правила принято называть интерфейсом. Интерфейс определяет последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на соседних уровнях в одном узле. Интерфейс определяет набор услуг, предоставляемый данным уровнем соседнему уровню. В сущности, протокол и интерфейс выражают одно и то же понятие, но традиционно в сетях за ними закреплены разные области действия: протоколы определяют правила взаимодействия модулей одного уровня в разных узлах, а интерфейсы — модулей соседних уровней в одном узле. Средства каждого уровня должны отрабатывать, во-первых, собственный протокол, а во-вторых, интерфейсы с соседними уровнями.

***Маршрутизация в компьютерных сетях: способы и методы.***

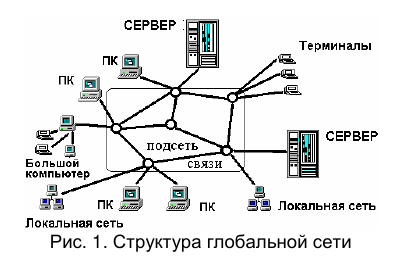
Различают следующие способы маршрутизации:

1. Централизованная маршрутизация. Реализуется обычно в сетях с централизованным управлением. Выбор маршрута для каждого пакета осуществляется в центре управления сетью, а узлы сети связи только воспринимают и реализуют результаты решения задачи маршрутизации. Такое управление маршрутизацией уязвимо к отказам в центральном узле и не отличается высокой гибкостью.
2. Распределенная (децентрализованная) маршрутизация. Выполняется главным образом в сетях с децентрализованным управлением. Функция управления маршрутизации распределены между узлами сети, которые располагают соответствующими средствами. Распределённая маршрутизация сложнее централизованной маршрутизации, но отличается большей гибкостью.
3. Смешенная маршрутизация. Характеризуется тем, что в ней в определённом соотношении реализованы принципы централизованной и распределённой маршрутизации.

Различают следующие методы маршрутизации:

1. Простая маршрутизация. Отличается тем, что при выборе маршрута не учитывается ни изменение топологии сетей, ни изменение её состояния (нагрузки). Она не обеспечивает направленной передачи пакетов и имеет низкую эффективность. Её преимущества - простая реализация алгоритма маршрутизации и обеспечение устойчивой работы сети при выходе из строя отдельных её элементов.
2. Случайная маршрутизация. Характеризуется тем, что для передачи пакета из узла связи выбирается одно, случайно выбранное свободное направление. Пакет «блуждает» по сети с конечной вероятностью когда-либо достигнуть адресата. Естественно при этом не обеспечивается ни оптимальное время доставки пакета, ни эффективное использование пропускной способности сети.
3. Лавинная маршрутизация. Предусматривает передачу пакета из узла по всем свободным выходным линиям. Поскольку это происходит в каждом узле, имеет место явное «размножение» пакета, что резко ухудшает использование пропускной способности сети. Значительное ослабление этого недостатка достигается путем уничтожения в каждом узле дубликатов пакета и продвижение по маршруту только одного пакета. Основные преимущества этого метода – гарантированное обеспечение оптимального времени доставки пакета адресату, так как из всех направлений передачи пакета хотя бы одно обеспечивает такое время. Метод может использоваться в незагруженных сетях, когда требования по минимизации времени и надежности доставки пакетов высоки.
4. Фиксированная маршрутизация характеризуется тем, что при выборе маршрута учитывается изменение топологии сети и не учитывается изменение её нагрузки. Для каждого узла назначение направления передачи выбирается по таблице маршрутов (каталогу), которая определяет кратчайшие пути. Каталоги составляются в центре управления сетью. Они составляются заново при изменении топологии сети. Отсутствие адаптации изменению нагрузки приводит к задержкам пакетов в сети.
5. Однопутевая маршрутизация. Строится на основе единственного пути передачи пакетов между двумя абонентами, что сопряжено с неустойчивостью к отказам и перегрузкам.
6. Многопутевая маршрутизация. Строится на основе нескольких возможных путей между двумя абонентами, из которых выбирается предпочтительный путь.
7. Адаптивная маршрутизация. Отличается тем, что принятие решение о направлении передачи пакетов осуществляется с учетом изменения как топологии, так и нагрузки сети.
8. Локальная маршрутизация. Основана на использовании информации, имеющейся в данном узле. Информация о состоянии других узлов не используется. Преимущество такого метода состоит в том, что принятие решения о выборе маршрута производится с использованием самых последних данных о состоянии узла. Недостаток метода в его близорукости, поскольку выбор маршрута определяется без учета глобального состояния всей сети. Следовательно, всегда есть опасность передачи пакета по перегруженному маршруту.
9. Распределенная маршрутизация. Основана на использовании информации, указанной для локальной маршрутизации, и данных, получаемых от соседних узлов в сети. В каждом узле формируется таблица маршрутов ко всем узлам назначения, где указываются маршруты с минимальным временем задержки пакетов. До начала работы это время оценивается исходя из топологии сети. В процессе работы сети узлы периодически обмениваются с соседними узлами таблицами задержки, в которых указывается нагрузка узла. После обмена информацией каждый узел пересчитывает задержки и корректирует маршруты с учетом поступивших данных.
10. Централизованная маршрутизация. Характеризуется тем, что задача маршрута для каждого узла в сети решается в центре маршрутизации. Каждый узел периодически формирует сообщение о своём состоянии (длине очередей и работоспособности линий связи) и передает его в центр маршрутизации. По этим данным для каждого узла составляют таблицы маршрутов.
11. Гибридная маршрутизация. Основана на использовании таблиц маршрутов, рассылаемых центром маршрутизации узлам в сети, в сочетании с анализом длины очередей в узлах. Гибридная маршрутизация компенсирует недостатки централизованной маршрутизации и локальной и воспринимает их преимущества.

***Глобальная компьютерная сеть Интернет.***

Глобальная компьютерная сеть объединяет компьютеры, удалённые друг от друга на большое расстояние и которые могут быть расположены в различных городах, государствах и континентах. Обмен информацией между компьютерами в такой сети может осуществляться при помощи телефонных линий связи, выделенных каналов связи, в том числе оптоволоконных, систем радиосвязи и спутниковой связи. В общем случае глобальная сеть включает подсеть связи, к которой подключены компьютеры и терминалы (только ввод и отображение данных). В состав глобальной сети могут входить как компоненты локальные и региональные сети. Объединение глобальных, региональных и локальных вычислительных сетей позволяет создавать многосетевые иерархии. Они обеспечивают мощные, экономически целесообразные средства обработки огромных информационных массивов и доступ к неограниченным информационным ресурсам. Именно такая структура принята в наиболее известной информационной сети Интернет. Подсеть связи состоит из каналов передачи данных и коммуникационных узлов. Компьютеры, за которыми работают пользователи-клиенты, называются рабочими станциями. Компьютеры, являющиеся источниками ресурсов сети, предоставляемых пользователям, называются серверами. Рабочие станции пользователей подключаются к глобальным сетям чаще всего через поставщиков услуг доступа к сети — провайдеров. Коммуникационные узлы подсети связи предназначены для быстрой передачи информации по сети, для выбора оптимального маршрута передачи информации, для коммутации пакетов передаваемой информации. Коммуникационный узел — это либо некоторое аппаратное устройство, либо компьютер, выполняющий заданные функции с помощью соответствующего программного обеспечения. Эти узлы обеспечивают эффективность функционирования сети связи в целом. Рассмотренная структура сети называется узловой и используется, прежде всего, в глобальных сетях. На примере всемирной глобальной сети Internet рассмотрим структуру глобальной сети. Сеть Internet представляет собой совокупность взаимосвязанных коммуникационных центров, к которым подключаются региональные поставщики сетевых услуг и через которые осуществляется их взаимодействие, т.е. Internet имеет типичную для глобальных сетей структуру.

***Основные принципы построения и управления сетью.***

Чаще всего в локальных сетях используются два основных типа передачи данных между компьютерами – по проводам, такие сети называются кабельными и используют технологию Ethernet, а также с помощью радиосигнала по беспроводным сетям, построенных на базе стандарта IEEE 802.11, более известный под названием Wi-Fi. На сегодняшний день проводные сети до сих пор обеспечивают самую высокую пропускную способность, позволяя пользователям обмениваться информацией со скоростью до 100 Мбит/c (12 Мб/c) или до 1 Гбит/с (128 Мб/с) в зависимости от используемого оборудования. Хотя современные беспроводные технологии чисто теоретически тоже могут обеспечить передачу данных до 1.3 Гбит/c, на практике эта цифра выглядит гораздо скромнее и в большинстве случаев не превышает величину 150 – 300 Мбит/с. Виной тому служит дороговизна высокоскоростного Wi-Fi оборудования и низкий уровень его использования в нынешних мобильных устройствах. Современные домашние сети устроены по одному принципу: компьютеры пользователей (рабочие станции), оборудованные сетевыми адаптерами, соединяются между собой через специальные коммутационные устройства, в качестве которых могут выступать: маршрутизаторы (роутеры), коммутаторы (хабы или свитчи), точки доступа или модемы. В самом начале необходимо определить основные требования к будущей сети и ее масштаб. Ведь от количества устройств, их физического размещения и возможных способов подключения, напрямую будет зависеть выбор необходимого оборудования. Чаще всего домашняя локальная сеть является комбинированной и в ее состав может входить сразу несколько типов коммутационных устройств.

Существует два основных принципа управления в локальных сетях:

В сетях с централизованным управлением функции управления обменом данными возложены на файл-серверы. Файлы, хранящиеся на сервере, доступны PC сети. Одна PC к файлам другой PC доступа не имеет. Обмен файлами между PC может происходить и в обход основных путей, например, с помощью программы NetLink. После запуска этой программы на двух компьютерах можно передавать файлы между ними. Преимуществом централизованных сетей является высокая защищенность сетевых ресурсов от несанкционированного доступа, удобство администрирования сети, возможность создания сетей с большим числом узлов. Основной недостаток состоит в уязвимости системы при нарушении работоспособности файл-сервера (преодолевается при наличии нескольких серверов), а также в предъявлении довольно высоких требований к ресурсам серверов.

Децентрализованные (одноранговые) сети не содержат в своем составе выделенных серверов. Функции управления сетью в них поочередно передаются от одной PC к другой. Ресурсы одной PC (диски, принтеры и другие устройства) оказываются доступными другим PC. Развертывание одноранговой сети для небольшого числа PC часто позволяет построить более эффективную и живучую распределенную вычислительную среду. Сетевое программное обеспечение в них является более простым по сравнению с централизованными сетями. Здесь не требуется установка файл-сервера (как компьютера, так и соответствующих программ), что существенно удешевляет систему. Однако такие сети слабее с точки зрения защиты информации и администрирования.

***Адресация в Интернет.***

Основным протоколом сети Интернет является сетевой протокол TCP/IP. Каждый компьютер, в сети TCP/IP (подключенный к сети Интернет), имеет свой уникальный IP-адрес или IP – номер. Адреса в Интернете могут быть представлены как последовательностью цифр, так и именем, построенным по определенным правилам. Компьютеры при пересылке информации используют цифровые адреса, а пользователи в работе с Интернетом используют в основном имена. Цифровые адреса в Интернете состоят из четырех чисел, каждое из которых не превышает двухсот пятидесяти шести. При записи числа отделяются точками. Для отдельного компьютера или локальной сети, которые впервые подключаются к сети Интернет, специальная организация, занимающейся администрированием доменных имен, присваивает IP – номера. Первоначально в сети Internet применялись IP – номера, но когда количество компьютеров в сети стало больше чем 1000, был принят метод связи имен и IP – номеров, который называется сервер имени домена (Domain Name Server, DNS). Сервер DNS поддерживает список имен локальных сетей и компьютеров и соответствующих им IP – номеров. В Интернете применяется так называемая доменная система имен. Каждый уровень в такой системе называется доменом. Типичное имя домена состоит из нескольких частей, расположенных в определенном порядке и разделенных точками. Домены отделяются друг от друга точками. В Интернете доменная система имен использует принцип последовательных уточнений. Домен верхнего уровня располагается в имени правее, а домен нижнего уровня – левее. Доменная система образования адресов гарантирует, что во всем Интернете больше не найдется другого компьютера с таким же адресом. Для доменов нижних уровней можно использовать любые адреса, но для доменов самого верхнего уровня существует соглашение. При работе в Internet используются не доменные имена, а универсальные указатели ресурсов, называемые URL (Universal Resource Locator). URL - это адрес любого ресурса (документа, файла) в Internet, он указывает, с помощью какого протокола следует к нему обращаться, какую программу следует запустить на сервере и к какому конкретному файлу следует обратиться на сервере. Общий вид URL: протокол://хост-компьютер/имя файла.

***Основные службы Интернет.***

Службы сети Internet - это системы, предоставляющие услуги пользователям сети. К ним относятся:

Электронная почта. Представляет собой систему, которая дает возможность формировать, пересылать, хранить и читать текстовые документы, электронные таблицы в определенном формате, графические файлы, программы и т.д. В электронном письме можно применять цифровой подписи и шифрования. Особенностью E-mail является то, что она действует в режиме offline.

Internet-пейджеры. Бесплатные программы для непосредственного общения в сети. Они могут выполнять функции передачи файлов, электронной почты, голосовых сообщений, новостей и т. п.

Телеконференция (форум). Услуга Internet, с помощью которой абонент может оставить свое открытое сообщение в сети. Позже его могут просмотреть заинтересованные пользователи и дать такое же открытое в ответ. Фактически, телеконференция представляет собой коллективную тематическую электронную почту.

Списки рассылок. Услуга, которая заключается в объединении по одному адресу электронной почты адресов многих подписчиков списка рассылки. Если на такой адрес высылается письмо, то его получают все подписчики этого списка. Эта служба не имеет собственного протокола и работает только через электронную почту.

WWW (World Wide Web). Служба прямого доступа, требующая полноценного подключения к Internet и позволяющая интерактивно взаимодействовать с представленной на Web-серверах информацией. Она базируется на принципе гипертекста и может представлять информацию с применением всех возможных мультимедийных средств: аудио, видео, текст, графика и т.п.

FTP-услуга. Служба прямого доступа к файлам в файловых Internet-архивах, которая требует полноценного подключения к сети.

IRC- система "разговорных комнат", так называемых чатов. Это интерактивная служба, которая дает возможность общения пользователей в режиме реального времени с помощью текстовых сообщений, которые набираются на клавиатуре.

Распространенными в сети Internet является службы поиска информации, которые поддерживаются одной из групп участников сети. Основными инструментами поиска информации являются:

- поисковые машины, основная функция которых заключается в исследовании Интернета с целью сбора данных о существующих Web-сайтах и выдача отчета о Web-страницах, удовлетворяющих требованиям запроса пользователя. Поисковая машина фактически является большой локальной сетью, которая состоит из мощных компьютеров с большим объемом дисковой памяти. Последние разделены на подгруппы, между которыми распределяется информация. На сервере поисковой машины текст разбивается на отдельные слова, каждому из которых присваиваются определенные координаты, которые заносятся в таблицу сервера вместе с гиперссылкой на IP-адрес. Когда поисковая система получает запрос, она ищет ответ в собственной таблице, а не в сети Internet.

- каталоги, иерархически организованные тематические структуры, в которых информация заносится по инициативе пользователей. Также страница, которая добавляется в каталог должна быть жестко привязана к определенной категории в каталоге. При этом поиск информации ведется не в сети Internet, а в компьютерных тематических базах данных;

- поисковые сайты, дают возможность пользователям проводить поиск информации по ключевым словам в больших базах данных Web-сайтов.

***Инструментальные средства разработки Web-сайтов.***

Для разработки web-страниц сайта можно использовать следующие инструментальные средства:

1. Текстовые редакторы для работы с “чистым” HTML-кодом. Поскольку HTML-документ представляет собой текстовый файл, его можно подготовить в простейшем текстовом редакторе, но при этом все команды разметки придется вводить вручную.
2. Программные средства, которые имеют специальные встроенные компоненты для конвертирования данных, созданных в этих продуктах, в HTML формат. Например, программы широко используемого пакета MS Office*,* позволяют сохранить документ в формате HTML. При этом сохраняются, насколько это возможно, особенности форматирования символов и абзацев, изображения, таблицы, списки и так далее. Многие современные графические редакторы предоставляют возможность сохранить коллекцию картинок в виде HTML-документа. Надо отметить, что при таком конвертировании исходный текст HTML-документа получается крайне избыточным и нуждается в коррекции.
3. Web-редакторы, предназначенные специально для разработки web-сайтов. Современные требования, предъявляемые к web-редакторам, включают в себя:

* Поддержку каскадных таблиц стилей.
* Использование современных скриптовых языков, таких как JavaScript и т.д.
* Генерацию Dynamic HTML для различных браузеров.
* Средства наглядного дизайна: вставка изображений, таблиц и фреймов.
* Динамическое отображение создаваемой страницы в браузере.
* Шаблоны WWW-страниц или специальные программы - "мастера" по их созданию.
* Средства по управлению web-сайтом (создание файловой структуры сайта, отслеживание связей и проверка ссылок между страницами, публикация сайта в Интернет).

Существуют две разновидности web-редакторов:

Невизуальные редакторы работают с "чистым" HTML-кодом. Они предоставляют возможность быстрого ввода тэгов HTML с использованием инструментальных панелей, вставки изображений и таблиц с помощью специальных диалоговых окон, работы с файловым менеджером для выбора папок и файлов, подсветки синтаксиса, проверки правильности получившегося кода и ссылок, использования справки по тегам и их атрибутам. Невизуальные web-редакторы часто называют HTML-редакторами.

Визуальные редакторы позволяют работать с самой web-страницей в режиме WYSIWYG (What You See Is What You Get). Редактирование и форматирование текстов, вставка рисунков, таблиц, гиперссылок и других элементов происходит как в обычном текстовом редакторе, а сама программа формирует соответствующий HTML-код. Кроме средств визуального редактирования эти web-редакторы одновременно предоставляют доступ к получившемуся HTML-коду.

***Языки HTML, XML.***

HTML (HyperText Markup Language) – язык разметки гипертекста – предназначен для написания гипертекстовых документов. Гипертекстовый документ – это текстовый файл, имеющий специальные метки, называемые тегами, которые впоследствии опознаются браузером и используются им для отображения содержимого файла на экране компьютера. С помощью этих меток можно выделять заголовки документа, изменять цвет, размер и начертание букв, вставлять графические изображения и таблицы. Основным преимуществом гипертекста перед обычным текстом является возможность добавления к содержимому документа гиперссылок – специальных конструкций языка HTML, которые позволяют щелчком мыши перейти к просмотру другого документа. HTML-документ содержит символьную информацию. Одна ее часть – собственно текст, т. е. данные, составляющие содержимое документа. Другая – теги, называемые также флагами разметки, – специальные конструкции языка HTML, используемые для разметки документа и управляющие его отображением. Именно теги языка HTML определяют, в каком виде будет представлен текст, какие его компоненты будут исполнять роль гипертекстовых ссылок, какие графические или мультимедийные объекты должны быть включены в документ. Графическая и звуковая информация, включаемая в HTML-документ, хранится в отдельных файлах. Программы просмотра HTML-документов интерпретируют флаги разметки и располагают текст и графику на экране соответствующим образом. Для файлов, содержащие HTML-документы, приняты расширения .htm или .html.

XML (eXtensible Markup Language) - это язык разметки, описывающий целый класс объектов данных, называемых XML- документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. Термин XML обозначает особую разновидность обычных текстовых документов. Фактически XML – это способ разметки документов, предназначенный для формирования в документах какой-либо структуры и определения отношений между различными элементами этой структуры. Для создания такой разметки служат специальные инструкции, называемые тегами. Благодаря наличию тегов становится возможной унифицированная автоматическая обработка и форматирование XML-документов. Таким образом, XML-документ может быть легко преобразован и отформатирован автоматически в соответствии с любыми требованиями любое количество раз. Благодаря XML удается также контролировать правильность данных, хранящихся в документах, а также установить единый стандарт на структуру документов, в которых могут содержаться произвольные данные. XML незаменим при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными потребителями. Благодаря XML можно избавиться от многих проблем, связанных с несовместимостью используемых различными компонентами системы форматов данных.

**Основные задачи защиты информации.**

***Основные понятия и проблемы защиты информации (угрозы, требования, критерии, способы, средства).***

Под угрозой безопасности для системы понимаются возможные воздействия, которые прямо или косвенно могут нанести ущерб ее безопасности. Основные типы угроз:

1. Угрозы конфиденциальности – несанкционированный доступ к данным;
2. Угрозы целостности – несанкционированная модификация, дополнение или уничтожение данных;
3. Угрозы доступности – ограничение или блокирование доступа к данным.

Основные требования. Защита информации должна быть:

1. Централизованной. Процесс управления всегда централизован, в то время как структура системы, реализующей этот процесс, должна соответствовать структуре защищаемого объекта;
2. Плановой. Планирование осуществляется для организации взаимодействия всех подразделений объекта в интересах реализации принятой политики безопасности; каждая служба, отдел, направление разрабатывают детальные планы защиты информации в сфере своей компетенции с учетом общей цели организации;
3. Конкретной и целенаправленной. Защите подлежат абсолютно конкретные информационной ресурсы, могущие представлять интерес для конкурентов;
4. Активной. Требование предполагает наличие в составе системы информационной безопасности средств прогнозирования, экспертных систем и других инструментариев, реализующих наряду с принципом «обнаружить и устранить» принцип «предвидеть и предотвратить»;
5. Надежной и универсальной. Методы и средства защиты должны надежно перекрывать каналы утечки информации и противодействовать способам несанкционированного доступа независимо от формы представления информации, языка ее выражения и закрепленном за ней видом носителя;
6. Нестандартной. Разнообразной по используемым средствам;
7. Открытой. Для изменения и дополнения мер обеспечения безопасности информации;
8. Экономически эффективной. Затраты на систему защиты не должны превышать размеры возможного ущерба.

Критерии защиты информации:

1. Политика безопасности. Должны существовать правила доступа субъектов (пользователь, процесс) к объектам (любой элемент компьютерной системы, доступ к которому может быть ограничен.
2. Метки. Используются в качестве исходной информации для осуществления процедур контроля доступа.
3. Идентификация и аутентификация. Все субъекты и объекты компьютерной сети должны иметь уникальные идентификаторы, которые должны быть защищены от несанкционированного доступа, модификации, уничтожения.
4. Регистрация и учет. Все события, происходящие в компьютерной системе, должны отслеживаться и регистрироваться в защищенном объекте, доступ к которому может быть разрешен только специальной группе пользователей.
5. Контроль корректности функционирования средств защиты. Все средства защиты должны быть независимы, но проверяться специальными техническими и программными средствами.
6. Непрерывность защиты. Защита должна быть постоянной и непрерывной в любом режиме функционирования компьютерной сети.

Способы обеспечения безопасности информации в ИС:

1. Препятствие. Физическое преграждение пути злоумышленнику к защищаемой информации.
2. Управление доступом. Регулирование использования информации и доступа к ней за счет системы идентификации пользователей, их опознавания, проверки полномочий и т.д.
3. Криптография. Шифрование информации с помощью специальных алгоритмов.
4. Противодействие атакам вредоносных программ. Использование внешних накопителей информации только от проверенных источников, антивирусных программ, брандмауэров, регулярное выполнение резервного копирования важных данных и т.д.
5. Регламентация. Создание условий по обработке, передаче и хранению информации, в наибольшей степени обеспечивающих ее защиту.
6. Принуждение. Установление правил по работе с информацией, нарушение которых карается материальной, административной или даже уголовной ответственностью.
7. Побуждение. Призыв к персоналу не нарушать установленные порядки по работе с информацией, т.к. это противоречит сложившимся моральным и этическим нормам.

Средства защиты информации:

1. Технические (аппаратные) средства – сигнализация, решетки на окнах, генераторы помех воспрепятствования передаче данных по радиоканалам, электронные ключи и т.д.
2. Программные средства – программы-шифровальщики данных, антивирусы, системы аутентификации пользователей и т.п.
3. Смешанные средства – комбинация аппаратных и программных средств.
4. Организационные средства – правила работы, регламенты, законодательные акты в сфере защиты информации, подготовка помещений с компьютерной техникой и прокладка сетевых кабелей с учетом требований по ограничению доступа к информации и пр.

***Методы построения систем защиты информации (аппаратные, программные, организационные и др. аспекты).***

Организационные (административные) меры защиты - это меры организационного характера, регламентирующие процессы функционирования системы обработки данных, использование ее ресурсов, деятельность персонала, а также порядок взаимодействия пользователей с системой таким образом, чтобы в наибольшей степени затруднить или исключить возможность реализации угроз безопасности. Они включают:

• мероприятия, осуществляемые при проектировании, строительстве и оборудовании вычислительных центров и других объектов систем обработки данных;

• мероприятия по разработке правил доступа пользователей к ресурсам системы (разработка политики безопасности);

• мероприятия, осуществляемые при подборе и подготовке персонала системы;

• организацию охраны и надежного пропускного режима;

• организацию учета, хранения, использования и уничтожения документов и носителей с информацией;

• распределение реквизитов разграничения доступа (паролей, ключей шифрования и т.п.);

• организацию явного и скрытого контроля за работой пользователей;

• мероприятия, осуществляемые при проектировании, разработке, ремонте и модификациях оборудования и программного обеспечения и т.п.

Физические меры защиты основаны на применении разного рода механических, электро- или электронно-механических устройств и сооружений, специально предназначенных для создания физических препятствий на возможных путях проникновения и доступа потенциальных нарушителей к компонентам системы и защищаемой информации, а также технических средств визуального наблюдения, связи и охранной сигнализации.

Технические (аппаратно-программные) меры защиты основаны на использовании различных электронных устройств и специальных программ, входящих в состав автоматизированной системы и выполняющих функции защиты.