1. Понятия алгоритм, программа, процесс. Дайте формальное определение алгоритма и поясните его свойства. Понятия примитива и языка программирования.(2)

Алгоритм – это последовательность действий, в результате выполнения которых получается решение поставленной задачи.

Программа является записью алгоритма разработанного для решения некоторой задачи на компьютере.

Процесс – это деятельность связанная с выполнением программы, что тоже самое с выполнением компьютерного алгоритма. Например, в современных компьютерах можно запустить несколько параллельных, взаимодействующих между собой процессов.

Эта проблема решается путем создания четко определенного набора составных блоков (кубиков), из которых могут конструироваться описания алгоритмов. Такие блоки называются примитивами.

То, что примитивам даются точные определения, устраняет многие проблемы неоднозначности и одновременно требуют одинакового уровня детализации для всех описываемых с их помощью алгоритмов.

Язык программирования это набор примитивов вместе с набором правил, устанавливающих, как эти примитивы могут комбинироваться для представления более сложных идей.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Способы описания алгоритма: синтаксис, семантика, нотации Бекуса, схемы Насе-Шнейдермана, что такое псевдокод (4)

• Каждый примитив состоит из двух частей:

• Синтаксической и семантической.

• Синтаксис – это символьное представление примитива.

• Семантика – это значение (содержание) примитива.

Синтаксис – это символьное представление примитива.

Семантика – это значение (содержание) примитива

Схемы Насе - Шнейдермана позволяют уменьшить громоздкость

Slide 20-21, pres. 32-33

Это обычно набор команд некоторого языка с ослабленными правилами.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Понятия теории составления алгоритмов и решения задач. Четыре этапа решения задач. Методы: аналогий, декомпозиции, стратегии «разделяй и властвуй», «динамического программирования». (30)

Создать алгоритм – это значит найти метод решения задачи. Поэтому, чтобы понять, как создаются алгоритмы необходимо понять процесс решения задач.

Теория решения задач – не является специфической только для компьютерных наук

Понять существо задачи; Вникнуть в самую суть. Попытаться сходу ее решить не начиная изучение литературы.

Разработать план решения задачи. (Предложить идею алгоритма); На этом этапе появляется активное отношение к изучению литературы, что более плодотворно, читаешь не впрок, а с определенной целью.

Выполнить план. (Сформулировать алгоритм и написать программу). Возможно, для начала получить грубое решение, не учитывающее всяких нюансов.

Оценить точность решения

Попробовать простейшие, предельные случаи. Выяснить ограничения. Анализ полученных результатов – тестирование

Чаще всего выручает поиск решения по аналогии с уже известным решением похожих; Поэтому надо много читать, как решаются те или иные задачи. Т.е. учиться.

Если задача оказалась оригинальной, тогда используем - Поэтапное уточнение: разбить задачу на ряд подзадач, которые в свою очередь разбиваются на более простые. Нисходящий метод (индукция): стратегии разделяй и властвуй, динамическое программирование.

Иногда надо просто взглянуть на задачу «трезво», или «с другой стороны» с общих позиций (дедукция).

crv

Задача разбивается на независимые подзадачи (части) меньшей сложности (размерности).

Каждая подзадача решается отдельно.

Из отдельных решений подзадач строится решение исходной задачи используя рекурсивный алгоритм.

Эффективность алгоритма при этом может в некоторых случаях повышаться, если на каждом шаге разбиение производится на части приблизительно одинаковой размерности.

Не всегда удается разбить задачу на независимые подзадачи.

Если же подзадачи оказываются зависимы, т.е. имеют общие под-подзадачи, то метод "разделяй и властвуй" делает лишнюю работу, решая некоторые подзадачи по нескольку раз, увеличивая тем самым трудоемкость алгоритма.

Рекурсивный алгоритм в этом случае будет неэффективен и лучше использовать итерационный

Sdp

задача погружается в семейство задач той же природы (другими словами, разбивается на зависимые (могут пересекаться) подзадачи);

каждая подзадача решается отдельно один раз и необходимые значения решений подзадач запоминаются, что позволяет не решать снова уже встречавшиеся ранее подзадачи;

для исходной задачи строится возвратное рекуррентное соотношение, связывающее между собой необходимые значения решений зависимых подзадач

Программируется итерационный алгоритм.

Стратегия метода динамического программирования это попытка свести рассматриваемую задачу к более простым, однако зависимым задачам и опираясь на полученные и сохраненные в массиве результаты решения более простых, решить и исходную задачу.

Эта стратегия за счет увеличения затрат памяти позволяет избавиться от повторного решения пересекающихся подзадач.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Понятие универсального алгоритмического языка. Приведите пример простейшего универсального языка программирования. Как на нем реализовать оператор присваивания y:=x не уничтожая x .(5)

Мы уже знаем, что основой всех языков являются машинные языки. Все остальное это надстройки, позволяющие облегчить программирование тех или иных задач

Построим простейший язык, который будет содержать необходимый набор данных и операторов

Современные языки высокого уровня оперируют с разнообразными структурами данных: числа, массивы, строки, ......, объекты.

Однако после всех трансляций, процессор манипулирует битовыми комбинациями (01).

Поэтому наш простейший язык манипулирует только данными типа: комбинация двоичных разрядов произвольной длины:

{01101010000101010000010}

имя переменной: обозначает такую комбинацию

a,cd,ab,e,….

3 оператора присваивания:

cleare <имя>; очистка строки

incr <имя>; увеличить на 1 (101®110)

decr <имя>; уменьшить на 1

(если перемен. = 0, то

дальнейшего изменения

не будет)

Один оператор цикла:

While <имя> not 0 do

<повторяемые операторы>

...

...

end;

Это знакомый нам оператор While с одним условием имя ? 0.

Присвоить переменной х значение 3 (011):

cleare x;

incr x, incr x, incr x;

Присвоить z значение х (z:=x):

clear z;

While x not 0 do;

Incr z;

decr x;

end;

при этом однако уничтожается х. Это можно легко исправить, введя промежуточную переменную y .

Присвоить z значение х не уничтожая х:

clear y; clear z;

While x not 0 do; присвоить у значение х

Incr y;

decr x;

end;

While y not 0 do; присвоить z и x значение y

Incr z;

Incr x;

decr y;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5. Понятие машины Тьюринга. Нарисуйте и объясните ее работу. Как записывается команда и программа. Роль МТ в теории компьютерных наук. Тезис Черча, теорема Геделя.(6)

Блок управления считывает и записывает символы на ленте с помощью головки.

Лента бесконечна в обоих направлениях, разделена на ячейки, в каждую может быть помещен символ из определенного набора (алфавита внешних символов).

В любой момент вычислений МТ находится в одном из возможных состояний (их количество конечно).

Имеется алфавит внешних символов (a1...al)

Имеется алфавит состояний {q1...qk}. Имеется состояние начало вычислений, состояние останов.

Вычисления МТ состоит из последовательности шагов (тактов), выполняемых блоком управления.

Команда МТ(qi aj ® qmamR) считывается символ aj находясь в состоянии qi из ячейки

запись символа am в ячейку

R – сдвиг влево, вправо, оставить на месте

изменение состояния qm®qn.

Программа представляет последовательность таких команд сообщающих что делать исходя из состояния и содержимого считываемой ячейки aj .

Будучи по своей природе абстрактной, МТ может быть воплощена в разнообразных формах.

Фактически, все современные универсальные ВМ являются машинами Тьюринга, за исключением конечности их памяти.

Подобное сходство совсем не случайно, ведь МТ представляет саму суть вычислительного процесса.

Поразительным результатом математических исследований (Гедель) является обнаружение существования такого класса функций, для которых не существует алгоритмического вычисления. Такие функции получили название невычислимых.

Понимание границы между вычислимыми и не вычислимыми функциями равнозначно пониманию ограничений свойственным любым компьютерам в принципе.

Если функция вычислима то ее можно вычислить с помощью некоторой МТ и наоборот. Если удается доказать, что с помощью МТ ее нельзя вычислить, то она невычислима в принципе.

Значение МТ для современной теории вычислительных систем состоит в предположении, что все вычислимые функции могут быть описаны в виде программы для некоторой МТ

Это является содержанием Тезиса Черча-Тьюринга.

Всегда ли принципиально разрешимая задача имеет практическое решение?

Под размерностью задачи обработки данных понимают число исходных данных в ее формальном описании (далее n). Трудоемкость растет с увеличением количества данных.

Временной сложностью алгоритма называется время, затрачиваемое алгоритмом как функция размерности задачи, которую он решает Tn=f(n)

Емкостной сложностью алгоритма называется объем памяти, требуемый для его реализации как функция размерности Бn=f(n)

Задача(алгоритм) считается тем более сложной, чем больше время тратит на ее расчет ВМ

Так задача линейного поиска считается более сложной O(n/2) чем поиск двоичный O(log2n)

Алгоритм сортировки пузырьком O(n2) более сложен, чем алгоритм слияния O(n?log2n), хотя затраты на программирование наоборот

Задача о рюкзаке O(2n) еще боле сложная чем сортировка. Практически неразрешимая при n>30

Класс полиномиальных задач обозначается P

Класс недетерминированных полиномиальных НП для которых почти доказана полиномиальность (комивояжера, задачи прямого перебора)

Эта гипотеза пока не доказана в виде теоремы, однако имеется много фактов в подтверждении ее справедливости.

Следствие ТЧ-Т: вычислительная мощность МТ выше, чем у любой возможной ВМ.

То есть МТ представляет теоретический предел возможностей ВМ существующих на практике.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6. Понятие рекурсии. Определение рекурсивного алгоритма. Какие задачи решаются рекурсивными методами. Разбиение задачи на подзадачи – элементарные, тривиальные. Как организуются условия окончания рекурсивного алгоритма. Приведите примеры. (7,28)

Рекурсивным называется описание объекта, частично состоящее и определяемое с помощью самого описываемого объекта. После того, как вы познакомитесь с рекурсией, вы обнаружите, что она встречается достаточно часто.

Рекуррентное уравнение называется правильным, если значения аргументов у любой из функций в правой части соотношения меньше значения аргументов у функций в левой части соотношения;

если аргументов несколько, то достаточно уменьшения одного из них.

F(n,x)=F(n-1,x)

U(n,x)=U(n-1,x-2) В общем виде рекурсивный алгоритм P можно выразить как алгоритм, состоящий из комбинации последовательности действий S и самого P:

P = S (a,b,…, P).

На каждом шаге n комбинация

Pn = S (an-1,bn-1,…, Pn-1).

представляет элементарную задачу в предположении, что an-1,bn-1,…, Pn-1 уже вычислены.

Такое определение называется индуктивным

Решать задачу рекурсивно

(т.е. составить рекурсивный алгоритм это значит разложить задачу на подзадачи, которые затем подобным образом (т.е. рекурсивно) разбить на еще меньшие подзадачи;

на определенном уровне подзадачи становятся настолько простыми, что могут быть решены тривиально;

когда решены все элементарные подзадачи, тогда подзадачи составленные из элементарных тоже будут выполнены.

Исходная задача окажется выполненной, когда будут выполнены все подзадачи ее образующие.

Алгоритм разбиения исходной задачи на подзадачи записывается в виде рекуррентного соотношения.

При выполнении рекурсивной подпрограммы вначале осуществляется многократный переход от некоторого текущего уровня (i=n) организации алгоритма к нижнему (или верхнему) уровню последовательно, до тех пор, пока, наконец не будет получено разбиение до тривиального решение задачи.

В вышеприведенных примерах: при i=0 или i=1.

При обратном переходе от полученного тривиального решения к решению для заданного i=n выполняется решение последовательности элементарных подзадач с нарастанием i (или убыванием)

При каждом переходе к следующему уровню (рекурсивной активации процедуры) происходит запоминание всех ее локальных и формальных параметров.

В результате после n-й активации в памяти будет находиться список из n+1 комплектов таких параметров.

Организован этот список таким образом, что комплект параметров, засланный в список последним, будет вызываться первым, и наоборот. Такая структура организации списка данных в памяти получила название стек.

Под организацию стека каждой программе системой Delphi выделяется определенный ограниченный ресурс оперативной памяти.

Подобно любым циклическим алгоритмам рекурсивные подпрограммы могут приводить к своеобразному зацикливанию, когда в процессе последовательного перехода к нижнему уровню не выполняется условие, обеспечивающее нахождение тривиальной подзадачи.

Обычно окончание реализуют след. образом:

Рекурсивный параметр n при каждом обращении уменьшают

n-1 или увеличивают n+1

проверяют условие достижения тривиальной задачи

if (n=Nmin) или if(n=Nmax).

Важно также сделать оценку максимальной глубины рекурсии чтобы убедиться, что она достаточно мала и не приводит к переполнению программного стека

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7. Постановка задачи оптимального выбора. Двоичное дерево решений. Метод полного перебора. Приведите общую процедуру полного перебора вариантов. Метод ветвей и границ –как реализуется проверка по стоимости, эвристические методы. (1,27)

Каждый узел в дереве интерпретируется как результат выполнения некоторой последовательности шагов решения задачи перебора вариантов.

Ветвь в дереве при этом соответствует принятию очередного решения (шагу), которое ведет к более полному решению. В игровых деревьях, например, каждая ветвь соответствует ходу игрока.

Листы (узлы у которых нет выходящих ветвей) представляют собой окончательные варианты решений.

Цель состоит в том, чтобы из всех возможных вариантов решений найти наилучший по заданному критерию путь от корня до листа при выполнении некоторых условий-ограничений.

Форма дерева, критерий оптимальности и условия-ограничения определяются конкретной задачей.

В процедуре Vbr(i,wt,oct) для более компактной реализации суммирования веса и стоимости введены дополнительно два формальных параметра – wt – суммарный вес текущей выборки и oct – общая ее стоимость.

Вначале , т.е. максимальная стоимость

всех элементов.

Затем при движении вдоль выбираемого пути, по мере не включения элементов из oct вычитается их стоимость.

Таким образом oct в каждом узле имеет смысл стоимости, которую еще можно достичь, двигаясь от данного узла до листа.

Критерием

<приемлемости, не включения>

т.е. возможности продолжения построения текущей выборки без данного элемента, будет то, что после данного исключения общая стоимость текущей выборки oct будет не меньше полученной до этого стоимости Cmax оптимальной выборки находящейся в Sopt:

oct-a[i].c>Cmax.

Ведь если выполняется это неравенство, то продолжение поиска, хотя он и может дать некоторое решение, не приведет к более оптимальному решению, чем то, что уже было достигнуто. Следовательно, дальнейший поиск на текущем пути бесполезен.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8. Что такое ключ записи и основное требование к нему. Опишите алгоритм линейного поиска с барьером. Какова его эффективность? (9,24)

Обычно запись содержит некое ключевое слово (ключ), по которому ее находят среди множества других аналогичных записей.

В зависимости от решаемой задачи ключом может, например, служить фамилия или учетный номер или адрес.

Основное требование к ключу в задачах поиска и сортировки состоит в том, чтобы операция проверки на равенство была корректной.

Поэтому, например, в качестве ключа не следует выбирать действительное число, т.к. из-за всегда возможной ошибки округления, поиск нужного ключа может оказаться безрезультатным, хотя этот ключ в массиве имеется.

Для применения более эффективных методов поиска значения ключей должны допускать упорядоченность (т.е. проверку на > или <).

Желательно, чтобы в массиве записей не было повторяющихся ключей.

используется когда нет никакой дополнительной информации о хранении разыскиваемых данных.

Он представляет собой последовательный перебор массива до обнаружения требуемого ключа или до конца, если ключ не обнаружен:

i:=1; n1:=n+1;

While (a[i].k<>x) and (i<n1) do i:=i+1;

if i=n1 then ’элемент не найден’

else ’элемент I’;

Видно, что на каждом шаге требуется увеличивать индекс и вычислять логическое выражение с двумя проверками Так как элемент х с равной вероятностью может быть как вначале, так и в конце массива, то среднее количество повторений в цикле равно n/2.

Ввиду того, что в приведенном алгоритме в каждом цикле две проверки, его эффективность (по к-ву проверок) пропорциональна количеству элементов n, что обозначается О(n).

А можно ли уменьшить здесь затраты на поиск?

Единственная возможность – попытаться упростить логическое выражение, ограничившись одной проверкой в цикле с помощью введения вспомогательного элемента - барьера, который предохраняет от перехода за пределы массива

a[n+1].k=x; i=1;

While a[i].k<>x do i:=i+1;

if i=n+1 then ‘элемент не найден’

else ‘элемент i’;

Эффективность этого алгоритма в два раза выше предыдущего, и оценивается как О(n/2).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9. Что такое ключ записи и основное требование к нему. Опишите алгоритм двоичного поиска и дайте оценку его эффективности. (8,3)

-//-

В этом алгоритме отсутствует внутри цикла проверка совпадения a[m].k=x.

На первый взгляд это кажется странным, однако тестирование показывает, что в среднем, выигрыш от уменьшения количества проверок превосходит потери от нескольких «лишних» вычислений до выполнения условия i=j.

Видно, что алгоритм двоичного поиска рекурсивен, т.е. более сложная задача сводится к последовательности простейших:

Poisk(a[1]..a[n])= if a[m].k<x then Poisk(a[1]..a[n/2])

else Poisk(a[n/2+1]..a[n])

Легко подсчитать, что эффективность этого алгоритма оценивается как

2m=n ? m=log2n

это обозначается как О(lоg2n).

Т.е. для n=128 log2128=7 128/2/7@9

этот метод в 9 раз эффективнее линейного поиска с барьером

для n=1024 – в 50 раз,

для n=106 – в 2500 раз.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10. Изложите алгоритмы трех основных методов прямой сортировки массива и охарактеризуйте их эффективность. (10,23)

Для i=1..n-1 выполняется следующий элементарный алгоритм:

начиная от n до i последовательно проверяем упорядоченность двух соседних элементов a[j].k и a[j-1].k, и если они не упорядочены, то меняем их местами.

В результате такого обмена минимальный элемент перемещается на место i

for i:=1 to n-1 do

for j:=n downto i do

if a[j-1].k>a[j].k then

begin

r:=a[j-1];

a[j-1]:=a[j];

a[j]:=r;

end;

Допустим, что массив a[1..n] разбит на две части

a1...ai-1, ai...an

– в первой части элементы упорядочены, во второй – нет.

При i=2 такое разбиение получается автоматически, т.к. массив из одного элемента тривиально упорядочен.

На каждом шаге i=2..n выполняем элементарный алгоритм:

берем очередной элемент r из неупорядоченной части и включаем его в «нужное» место упорядоченной части.

a1...ai-1, ai...an

Для поиска нужного места в нижеприведенном алгоритме используется барьер a[0].k:=x.

Элементы, начиная от i-1 до нужного j сдвигаются на одну позицию: a[j]:=a[j-1]

for i:=2 to n do

begin

r:=a[i]; //элемент из неупор части

x:=a[i].k;//ключ этого элемента

a[0].k:=x; //барьер

j:=i;

while x<a[j-1].k do //поиск места

begin

a[j]:=a[j-1]; //сдвиг

j:=j-1;

end;

a[j]:=r;//вставка в нужное место

end;

Для i=1..n-1 выполняется следующий элементарный алгоритм: среди элементов ai...an выбираем минимальный am и переставляем местами элементы i-й и m-й.

a1...ai-1, ai…am...an

В результате на первое место станет самый минимальный, на второе – следующий минимальный и т.д

for i=1 to n-1 do

begin

m:=i; //поиск m

for j=i+1 to n do

if a[j].k<a[m].k then m:=j;

r:=a[i]; //перестановка

a[i]:=a[m];

a[m]:=r;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11. Изложите алгоритм метода быстрой сортировки Хоара. (11,22)

Выберем значение ключа среднего m-го элемента x=a[m].k (m можно выбирать либо делением пополам, либо случайным образом, как рекомендует автор).

i=1 j=n

Будем просматривать массив слева до тех пор пока не обнаружим элемент a[i].k>x.

После этого будем просматривать массив справа, пока не обнаружим a[j].k<x.

Поменяем местами элементы a[i] и a[j] и продолжим такой процесс просмотра (слева и справа, обмен) пока оба просмотра не встретятся где-то внутри массива (j<i причем j=i-1).

В результате массив окажется разбитым на

левую часть a[1..j], с ключами меньше (или равными) x

правую a[i..n], i=j+1, с ключами больше (или равными) x.

Var a:mas;n:word;

Procedure QuickSortR;

Procedure sort(L,R: word);

Var i,j:Word; w:Tzp; x:Tk;

begin

i:=L;j=R;

x:=a[(L+R) div 2].k;

Repeat

While a[i].k<x do i:=i+1;

While a[j].k>x do j:=j-1;

if i<=j then begin

w:=a[i];a[i]:=a[j];a[j]:=w;i=i+1;j=j-1;

end;

until i>j;

if L<j then sort(L,j);

if i<R then sort(i,R);

end; //sort

begin

sort(1,n);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12. Изложите алгоритм метода сортировки слиянием массива записей. Его эффективность (12)

Var a:mas; n: word;

Procedure SortSlip;

Procedure SrSL(L,R:word);

var m:word;

begin

if L<>R then begin

m:=(L+R) div 2;//разбиение

SrSL(L,m);

SrSL(m+1,R);

Slip(L,m,R);

end;

end;

begin

SrSL(1,n)

end; var a:mas;

Procedure Slip(L,m,R:word);

Var c: mas; i,j,k:word;

begin

i:=L; k=1; j:=m+1;

while (i<=m) and (j<=R) do

if a[i].k<a[j].k

then begin c[k]:=a[i]; Inc(i); Inc(k) end

else begin c[k]:=a[j]; Inc(j); Inc(k) end;

while i<=m do

begin c[k]:=a[i]; Inc(i); inc(k) end;

while j<=R do

begin c[k]:=a[j]; Inc(j); Inc(k) end;

k=0;

for i:=L to R do

begin Inc(k); a[i]:=c[k] end;

end;

является иллюстрацией того, каким образом за счет использования дополнительных рабочих массивов можно значительно повысить эффективность сортировки.

Идея метода основана на следующем эффективном алгоритме слияния.

Пусть внутри массива a[1..n] имеется два смежных отсортированных участка (например, по возрастанию ключа) a[L, m], a[m+1,R] 1?L<R?n.

a1….aL...am, am+1...aR….an

Необходимо отсортировать весь участок a[L,R].

Процедура сортировки для этого участка организуется специфическим слиянием двух участков в один рабочий массив

C1…CR-L+1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13. Напишите алгоритмы и дайте сравнительную характеристику эффективности методов прямой и быстрой сортировок обменом элементов. Что такое эффективность метода и как она оценивается. (26)

О(n2) I log2n

Эффективность сортировки

Методы внутренней сортировки классифицируются по времени их работы.

Хорошей мерой эффективности может быть

число сравнений ключей - С

число пересылок элементов - Р.

Эти числа являются функциями С(n), Р(n) от числа сортируемых элементов n.

Быстрые (но сложные) алгоритмы сортировки требуют

при n®? порядка n?lgn сравнений т.е. их эффективность O(n?lgn)

Прямые (простые) методы - O(n2).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

14. Что такое список? Как организуется список на основе динамического массива данных. Понятие стека и очереди. Методы добавления и чтения элемента. (13, 17)

В практике программирования довольно часто встречается задача обработки набора однотипных данных

Inf1, Inf2, ...... Infn,

количество которых n меняется в зависимости от ситуации

Их надо разместить в оперативной памяти и по мере надобности с ними работать: добавлять новые данные в список, освобождаться от старых данных, выдавать информацию о количестве данных, находить нужную информацию

Такой набор программисты называют СПИСКОМ

Простейшая форма организации списка – это динамический массив данных. Данные, размещенные в массиве имеют свои номера (индексы) и это придает эффект «видимости» каждому элементу.

Turn-это список с двумя точками входа. С одной стороны последовательности данные добавляются в список (в конец очереди), с другой стороны списка данные удаляются из него (из начала очереди)

Таким образом реализуется принцип – «первым пришел – первым вышел»

Наглядный пример – трубка, заполняемая шариками, имеющими диаметр, равный диаметру трубки. С одной стороны трубки шарики добавляются в нее, с другой вынимаются.

Stack- это список с одной точкой входа.

Данные добавляются в список и удаляются из него только с одной стороны последовательности (вершины стека)

Таким образом реализуется принцип – «последний пришел – первым вышел»

procedure Tlist.Addk(c:Tinf);

begin

GetMem(a1,(n+1)\* mt);

a1[n+1]:=с; //добавление элемента

if n>0 then begin

for i:=1 to n do // копирование

a1[i]:=a[i];

FreeMem(a,mt\*n);//освобождение

end;

a:=a1; n:=n+1; //теперь имя опять a

end;

procedure Tlist.Read1(var c:Tinf);

begin

if n>0 then begin

c:=a[1]; n:=n-1;

if n>0 then begin

GetMem(a1,n\* mt);

//копирование

for i:=1 to n do a1[i]:=a[i+1];

end else a1:=nil;

FreeMem(a,mt\*(n+1)); //освобождение

a:=a1; //теперь имя опять a

end else «список пуст!!!»;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15. Что такое список? Как организуется однонаправленный список на основе рекурсивного типа данных. Понятие стека и очереди. Методы добавления и чтения элемента. Что такое однонаправленный список с меткой? Как организуется циклический и двунаправленный списки. (14,20))

Здесь используется единственное исключение языка Паскаль – тип (sel) используется раньше, чем он описан

Как видим, описание типа Tsel содержит внутри обращение к самому себе (A:Tsel), т.е. оно рекурсивно.

При этом А является указателем на ячейку памяти точно такой же структуры.

В переменной Inf размещаются данные о некотором элементе списка, причем по крайней мере в одном из полей Inf, а иногда и в отдельном поле записи расположены сведения по которым производится поиск требуемой информации.

Это поле будем обозначать key:Tkey, а сведения называть ключом.

Программисту известен только один указатель sp1, который назовем вершиной стека

Доступ к ячейкам стека возможен только через этот указатель, в котором расположен адрес первой ячейки.

Программисту известны два указателя sp1, spk

Доступ к ячейкам стека возможен только через эти указатели, в которых расположены адрес первой и адрес последней ячеек

В конец очереди через spk добавляют новые ячейки, из начала очереди через sp1 удаляют ячейки.

Методы работы с очередью немного отличаются от методов работы со стеком – при работе с очередью необходимо поддерживать вторую точку входа spk

Поэтому более удобно написать два класса – один для работы со стеком, другой с очередью

Как мы видели, в однонаправленном списке довольно сложно взять элемент изнутри списка

Например последний элемент из очереди просто так удалить не получается, хотя имеется его адрес spk

Так же проблематично вставить элемент внутрь очереди перед заданным

Хотя довольно эффективно работают процедуры удаления и вставки элемента после элемента с заданным адресом ReadAftter, AddAftter.

Однако при их регулярном использовании имеется проблема первого элемента – его надо обрабатывать другой процедурой, а значит каждый раз требуется лишняя проверка

Чтобы избавиться от нее, часто используют Список с меткой

В таком списке от первой ячейки используется только ее адресная часть sp1.A,

в информационной же части могут быть при необходимости размещены данные об этом списке, при этом метка создается методом Add1

Stekm:=create; Stecm.add1(Infm); .

При работе со стеком sp1 остается неизменным

После создания списка с меткой работа с ним производится с помощью единообразных процедур ReadAfter, AddAfter, PoiskAfter.

Циклические списки используются, когда нужно обходить набор элементов в бесконечном цикле.

Они так же позволяют получить доступ к списку начиная с любой позиции, что придает списку некоторую симметрию.

Программа может работать со всеми элементами одинаково без использования меток с помощью процедур RedAfter, AddAfter, PoiskAfter.

организуются когда требуется просматривать список как в одном так и в обратном направлениях.

Мы видели, что в однонаправленном списке довольно просто удалить (вставить) новую ячейку после заданной, но довольно сложно удалить саму заданную ячейку или предыдущую к ней.

Эта проблема легко решается, если ввести рекурсивный тип с двумя адресными ячейками

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

16. Для чего нужны древовидные структуры данных? Приведите и проиллюстрируйте основные определения: порядок узла, степень дерева, глубина, сбалансированность. Напишите класс и методы обхода дерева и его удаления. (15, 25)

При решении игровых задач мы встретились с понятием дерева, которое использовалось в качестве математической модели для представления множества возможных вариантов.

Познакомились с рекурсивным алгоритмом обхода всех узлов и листьев дерева и построенным на нем методом ветвей и границ.

Древовидная модель оказывается довольно эффективной и для представления динамических данных с целью быстрого поиска информации.

Если два узла соединены направленной дугой (x®y) то

х - предшественник (родитель),

у – преемник (сын, дочь).

Корень - узел, у которого нет родителя.

Лист - Узел, не имеющий преемников

Внутренний узел – это узел, не являющийся листом или корнем.

Порядок узла – количество его дочерних узлов.

Степень дерева – максимальный порядок его узлов

Глубина узла равна числу его предков плюс 1.

Глубина дерева – это наибольшая глубина его узлов.

Дерево, представленное выше имеет степень 3 (троичное), глубину 4, корневой узел а, листы d, e, g, k,m

Procedure DeleteTree(var p:Ttree);

Begin

If (p<>Nil) then begin

DeleteTree(p^.A1);

DeleteTree(p^.A2);

. . . . .

DeleteTree(p^.Am);

Dispose(p);

p:=Nil;

end;

Ввиду своеобразной организации эффективность поиска информации в такой динамической структуре данных сравнима с эффективностью двоичного поиска в массиве, т.е. имеет оценку эффективности О(log2n).

Заметим что двоичный поиск в линейном связанном списке организовать невозможно, а эффективность линейного поиска имеет порядок О(n/2).

Конечно, оценка О(log2n) справедлива для сбалансированного дерева, т.е. такого у которого узлы, имеющие только одну дочь, имеют глубину равную или на единицу меньшую глубины дерева.

Begin

Del(proot); inherited free;

end;

Procedure TtreeB.Wrt1;

Procedure Wr(p:Ttree);

begin

if p<>nil then

begin

Wr(p^.A1);

Print(p^.Inf);

Wr(p^.A2);

end;

end;

begin

p:=proot; wr(p)

end;

Function TtreeB.poisk(k:Tkey):Tinf;

begin p:=Proot;

While(p<>nil) and (p^.Inf.key<>k) do

If k<p^.Inf.key then p:=p^.A1

else p:=p^.A2;

if p<>Nil then Result:=p^.inf

else Result.key:=nok;

end;

Обращение к методу

Inf:=tr.poisk(k);

If inf.key=nok then print(’нет ключа’)

else . . .

Procedure Ttree.Add(Inf:TInf);

Var bl:Boolean;

begin

New(w); // создание нового листа:

w^.Inf:=Inf;

w^.A1:=Nil;

w^.A2:=Nil;

if proot=Nil then proot:=w

else

begin

p:=proot;

repeat

q:=p; // q - указатель на предыдущий элемент

bl:=Inf.key<p^.Inf.key;

if bl then p:=p^.A1

else p:=p^.A2;

until p=Nil; // добавление листа

if bl then q^.A1:=w

else q^.A2:=w;

end;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

17. Что такое дерево поиска? Приведите пример идеально сбалансированного и плохо сбалансированного дерева поиска. Процедуры обхода и нахождения элемента в дереве поиска. (16,19)

ключи расположены таким образом, что для любого узла значение ключа у родителя больше чем у левого преемника, при этом значение ключа у левого преемника меньше чем у правого.

Ввиду своеобразной организации эффективность поиска информации в такой динамической структуре данных сравнима с эффективностью двоичного поиска в массиве, т.е. имеет оценку эффективности О(log2n).

Заметим что двоичный поиск в линейном связанном списке организовать невозможно, а эффективность линейного поиска имеет порядок О(n/2).

Конечно, оценка О(log2n) справедлива для сбалансированного дерева, т.е. такого у которого узлы, имеющие только одну дочь, имеют глубину равную или на единицу меньшую глубины дерева.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

18. Что такое хеширование? Понятие хеш-функции, хеш-таблицы, алгоритм разрешения конфликтов. Напишите класс для хеширования. Опишите три способа хеширования, которые вы изучали. (21)

Хеширование (hashing-перемешивание) состоит в том, что ключи данных записываются в так называемую хеш-таблицу (массив).

Идея в том, что при помощи некой простой функции i=h(key) алгоритм хеширования определяет положение искомого элемента в таблице по значению его ключа.

Допустим, имеется набор n записей с ключами в диапазоне 0?key?K, n<K.

Создадим массив (хеш-таблицу):

H:array[0..K] of <тип записей>;

Вначале его очистим H[i]:=0 и наши n записей помещаем в этот массив в соответствии со значением ключа i=key. Затем, используя операторы

zp:=H[key]; //извлекаем из массива

H[key]:=zp; //записываем в массив

Функция h(key) называется функцией расстановки или хеш-функцией.

Ввиду того, что число возможных значений ключа K обычно значительно превосходит возможное количество записей K>>n, любая функция расстановки может для нескольких значений ключа давать одинаковое значение позиции i в таблице.

Например, если M=10 то для key=1, key=11 и key=101 позиция i=h(key)=1.

В этом случае h(key) только определяет позицию, начиная с которой нужно искать запись с ключом key Ввиду такой неоднозначности схема хеширования должна включать алгоритм разрешения конфликтов, определяющий порядок действий, если позиция i=h(key) оказывается занятой записью с другим ключом.

Имеется множество схем хеширования различающихся как выбором удачной функции h(key) так и алгоритма разрешения конфликтов.

Эффективность решения реальной практической задачи будет существенно зависеть от выбираемой стратегии разрешения конфликтов.

содержит следующие действия:

сначала ключ key очередной записи отображается на позицию i=h(key) таблицы

если позиция свободна то в нее размещается элемент с ключом key,

если занята то отрабатывается алгоритм разрешения конфликтов, который находит место для размещения данного элемента.

Один из самых изящных способов разрешения конфликтов состоит в том, чтобы сохранять записи, отображаемые на одну позицию в связанные стеки. Для этого вводится массив из М указателей на связанные стеки

Можно и на массив бинарных деревьев поиска

Обычно выбирают М - простое число, немного большее чем предполагаемое количество записей.

Хеш-функция выбирается в виде i:=Key mod M.

Работу с таблицей удобно организовать посредством класса ТН

Преимущество рассмотренного метода заключается в том, что связанные хэш-таблицы никогда не переполняются, довольно просто осуществляется вставка, удаление и поиск элементов.

Недостаток таких таблиц в том, что если данные недостаточно равномерно перемешаны по ключу, то некоторые списки могут оказаться очень длинными в то время как большинство других будут пустыми, при этом поиск будет замедляться.

Для избавления от этого недостатка нужно придумать другую функцию хеширования.

Иногда используют следующую функцию

i:=(Key+P) Mod M,

где Р - простое число.

Можно Р генерировать с помощью датчика случайных чисел, при этом вначале записи и поиска датчик устанавливается в одинаковое начальное положение

Перемешанная таблица Этод метод послужил поводом, для названия (хеширование).

Используется простой массив H[i], функция хеширования i=Key mod M.

Алгоритм разрешения конфликтов следующий:

Если очередная позиция i занята, тогда ищется первая свободная позиция и туда помещается элемент. Для уменьшения эффекта кластеризации можно использовать функцию хеширования с датчиком случайных чисел, описанную выше.

Метод блоков Используется массив одномерных массивов (блоков)

Type Tblok=array[1..P] of Tinf;

Var H:array[0..M-1] of Tblok;

H1,H2:array[0..M-1] of Tblok;

блоки переполнения

Размещаются элементы следующим образом:

Вначале обычная функция i:=Key mod M находит номер блока и размещает в H[i] элемент (в конце блока).

Если i-й блок переполнен, то элемент помещается в очередной блок переполнения H1[i].

Этот способ обычно используется при хранении данных на файле. Запись и чтение в (из) файла осуществляется поблочно. Это позволяет значительно ускорить поиск, т.к. чтение из файла по одной записи медленнее, чем сразу блоком.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

19. Работа с разреженными матрицами на основе массива из стеков. Напишите класс, включающий конструктор и методы добавления и чтения элемента в разреженную матрицу (18)

Матрицы, содержащие большое число нулевых элементов называются разреженными

Решение многих прикладных задач (методы теории графов, численные методы решения дифференциальных уравнений методом конечных элементов, методы обработки изображений, методы криптографии) связано с обработкой матриц большой размерности n имеющих малое число ненулевых элементов

Использование специальных способов размещения в памяти разреженных матриц и разработка специальных алгоритмов работы с ними позволяет во многих случаях существенно снизить затраты компьютерных ресурсов и уменьшить время вычислений.

Эффективность работы с таким массивом определяется введением класса с продуманным набором методов и создание на этой основе эффективных процедур работы с матрицами

constructor TListms.create(m0,n0:word);

Var i:word;

begin m:=m0; n:=n0;

inherited create;

GetMem(ms,4\*m);

for i:=1 to m do begin

ms[i]:=Tlists.create;

ms[i].sp1:=nil

end;

end;

Procedure TListms.add(i,j:word;a:extended);

Var inf:Tinf;

begin

inf.a:=a; inf.j:=j;

ms[i].adds(inf);

end;

Function TListms.read(i,j):extended;

Var inf:Tinf;

begin

result:=ms[i].readsj(j);

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20. Понятия статического и динамического распределения памяти. Что такое указатель? Операции над указателями. Что такое динамическая переменная? Наложение переменных с помощью указателя

Все команды и данные программы во время ее выполнения размещаются в определенных ячейках оперативной памяти

При этом часть данных размещается в ячейки памяти, отводимой под программу еще на этапе компиляции и в процессе работы программы их адреса относительно начала программы не изменяются.

Такое размещение данных и команд называется статическим

Соответствующие этим данным переменные называются статическими переменными Возможна также организация динамического размещения данных

В этом случае под некоторые данные и программы память выделяется непосредственно во время выполнения по мере надобности, а после решения требуемой задачи память освобождается для других данных

Соответствующие таким данным переменные называются динамическими переменными

Для организации динамического распределения памяти используются переменные специального типа – указатели, которые обеспечивают работу непосредственно с адресами ячеек памяти.

Под каждую переменную типа указатель отводится ячейка объемом 4 байта, в которой помещается адрес нужной переменной Type Pk=^тип;

Pmi=^array[1..20] of integer;

Pb=^byte;

Var u,v:тип;

p,q:pointer;

a,b:Pk;

mi:Pmi

k:Pb;

s:^char;

Здесь р,q – нетипизированные указатели;

а,b,mi,k,s – типизированные указатели, т.е. они указывают, что, например, в ячейках начиная с адреса а размещается переменная указанного типа <тип>

Адрес статической переменной указателю можно присвоить следующим образом Если v, u – обычные статические переменные, то их адреса (адреса первого байта) можно получить с помощью специальной функции

p:=Addr(u); a:=Addr(v);

Очистка адреса из указателя осуществляется с помощью специальной функции Nil:

p:=Nil; a:=Nil,

довольно часто используется проверка условия

if p<>Nil then ...

if a=Nil then ... Значениями указателей являются адреса переменных, размещенных в памяти.

Значение одного указателя можно передать другому с помощью оператора присваивания, например:

p:=q;

a:=b;

При этом надо помнить, что в операторе присваивания типизированные указатели должны быть одного типа.

Используя нетипизированные указатели, можно передать адрес между указателями разного типа, например так:

p:=i; k:=p; Использование указателей позволяет «накладывать» переменные разных типов друг на друга, интерпретируя по-разному данные, расположенные по некоторому адресу

В этом случае мы имеем возможность неявного преобразования типов

Var Ch:Char;

i:^byte;

begin

i:=Addr(ch);

ch:=’A’;

Write(‘код ’,ch,’ = ’,i^:4);

Будет выведено сообщение

код А = 160

Процедура

New(a:<типизированный указатель>);

находит в куче свободный участок памяти, размер которого позволяет разместить тип данных а и присваивает указателю а значение адреса первого байта этого участка.

После этого данный участок памяти закрепляется за программой и с ним можно работать через возникшую в программе переменную a^.

Такие переменные называются динамическими.

После того как необходимость работы с этой переменной отпала, данный участок памяти освобождается с помощью процедуры

Dispose(a);

При этом адрес остается в указателе a

При работе как с типизированными так и с нетипизированными указателями аналогичные New, Dispose действия выполняют процедуры

GetMem(P:pointer;size:Word);

FreeMem(P,size);

здесь size - количество байт выделяемой памяти начиная с адреса, помещаемого в указатель Р.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

21. Понятие подпрограммы, для чего они нужны? Как описывается подпрограмма в виде Procedure? В виде Function? Что такое формальные и фактические параметры? В чём различие параметров-значений и параметров-переменных? Как вызвать процедуру? Функцию? Как оформить набор подпрограмм в виде библиотечного модуля (Unit)?

Подпрограмма – это последовательность операторов, оформленная таким образом, что ее можно вызвать по имени из любой программы, которую вы составляете, передать ей определенные параметры и получив требуемый результат продолжить вычисления. Вызов процедуры

…….

имяП(<список фактических параметров>);

Imap(x,y,m,k,j,S);

Вызов функции

…….

Y:=имяФ(<список фактических параметров>);

s:=y\*имяФ(сп.факт.пар.1)+y/имяФ(сп.факт.пар.2);

Z:=Imaf(a,b,m,k,j,S);

Y должно иметь типФ

Для каждого параметра – значения транслятор внутри подпрограммы резервирует дополнительные ячейки памяти в соответствии с типом параметра.

При вызове подпрограммы, происходит пересылка фактического параметра в эти ячейки памяти, после чего выполняется подпрограмма.

Значение ячейки, где находился сам фактический параметр не изменяется.

Этот механизм обеспечивает защищенность фактического параметра.

Фактическим параметром может быть константа, переменная или арифметическое выражение

Переменная

Фактическим параметром может быть только имя переменной.

При вызове подпрограмм передается адресс ячейки переменной, в которой находится фактический параметр, и все действия производятся над этой ячейкой.

После работы подпрограммы в ячейке фактического параметра при необходимости будет находиться результат.

Набор подпрограмм которые могут быть использованы при разработке целого ряда программ удобно оформить в виде отдельных тематических библиотек.

Для организации таких библиотек в Паскале введены модули.

Модуль (Unit) – это автономно компилируемая программная единица.

Модуль состоит из интерфейсной части, доступной для других программ, и исполняемой части которая скрыта.

В интерфейсной части располагаются необходимые константы, переменные и все заголовки подпрограмм помещенных в данную библиотеку.

В исполняемой части располагаются описания констант, переменных и подпрограмм.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

22.Как выводить графики в Delphi, используя компоненту Tchart? Опишите последовательность действий по настройке Tchart и фрагмент программы вывода графика y=sinx.

1) на закладке 3D Отключаем ^ 3Dimention

2) на закладке Panel

Выбираем Panel Color

3) на закладке Titles

Пишем Мой график

4) Нажимаем Add

Добавляем новый график

После нажатия Add

Вам предлагается выбрать вид графика.

Выбираем Line -

график в виде ломанной линии соедaиняющей последовательность точек с координатами (Xi,Yi)

После выбора двух графиков нажмем закладку Series

На закладке Border выберем толщину Width и стиль Style линий графика

На закладке Color выберем цвет линии

procedure TForm1.Button1Click(Sender: TObject);

var h,x,y,z:extended;

i:word;

begin

Chart1.SeriesList[0].Clear;

Chart1.SeriesList[1].Clear;

h:=0.1;

x:=0;

for i:=1 to 100 do begin

y:=sin(x);

z:=1/(x+1)

Chart1.SeriesList[0].AddXY(x,y);

Chart1.SeriesList[1].AddXY(x,z);

x:=x+h;

end;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

23. Как выводить графики, используя компоненту TImage Опишите последовательность действий по настройке TImag и фрагмент программы вывода закрашенного квадрата внутри круга

Любое изображение на экране дисплея формируется из набора светящихся точек, получивших название пиксели.

Процесс отображения осуществляется с помощью специальной микросхемы называемой графическим адаптером.

Работа графического адаптера осуществляется под управлением специальной программы - драйвера.

Адаптер содержит порты ввода-вывода информации, оперативную память в которой помещается таблица, содержащая информацию о каждом светящемся на экране пикселе (его координате, цвете, яркости). Рисование картинки в системе Delphi происходит в рамках создаваемого проекта на форме.

Наиболее удобно для этого пользовать компоненты Image (Addition)или Paintbox (Sistem). Можно рисовать непосредстввенно на форме.

Компоненты Image и PaintBox представляют собой выделенный на форме прямоугольник с канвой, на которую можно наносить изображения. Размеры прямоугольника в пикселях определяются на этапе проектирования.

Основное достоинство Image – это возможность записывать и считывать нарисованную картинку в файл.

Класс TCanvas

предоставляет в ваше распоряжение таблицу пикселей с координатами

pxmax:=Image1.Width; pymax:=Image1.Height;

Вместе с набором графических инструментов

Рисование начинается с очистки фона окна:

Image1.Picture:=Nil;

или

Image1.Canvas.Brush.color:=clWhite;

Image1.Canvas.Rectangle(0,0,pxmax,pymax);

Оператор - Установить цвет выбранного пикселя

Image1.Canvas.Pixels[px, py]:=ClBlack;

Можно начинать рисовать по точкам:

With Image1 do begin

dp:=Height div 4;

Px0:=Width div 2;

Py0:=Height div 2;

For px:=px0-dp to px0+dp do

canvas.pixels[px,py0]:=ClRed;

For py:=py0-dp to py0+dp do

canvas.pixels[px0,py]:=ClNavy;

End;

вычертится красно-синий крест. Он будет нарисован довольно тонккими линиями, т.к. рисуется по точкам.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

24. Что такое записи в языке Pascal? Как они описываются, в чем особенность вариантной части? Приведите примеры использования вариантной части.

Запись – это структура данных, в которой под одним именем объединены переменные, в общем случае разного типа, которые называются полями записи

Type

Tzap=record

a,b,c:Typ1; Разделы фиксированной части

e,f:Typ2; записи

........

z,x,y,u:Typn;

case byte of

1:(d:Typ01; Разделы вариантной части

g:Typ02); записи

........

m:(p,q:Typ1m) <-вариант m

end;

Var

Z1,Z2:Tzap;

Возможны записи, имеющие только фиксированную часть (отсутствует вариантная часть) или имеющие только вариантную часть (отсутствует фиксированная часть). Вариантная часть только одна и ставится в конце записи.

Каждому полю фиксированной части отводится ячейка памяти в соответствии с его типом.

Для вариантной части под все варианты полей отводится одна и та же область памяти, объем которой равен максимальному из вариантов полей.

Поэтому соответствующие поля разных вариантов располагаются на одних и тех же байтах памяти!

В вариантной части недопустимы длинные строки (string)!

Type

Tzp3=record

Case byte of

1:(a:array[0..255] of byte);

2:(b:array[0..255] of char);

end;

........

Var z3:Tzp;

begin

With z3 do begin

for i:=0 to 255 do

begin a[i]:=i; Writeln(b[i],a[i]);end;

end;

end;

Будет выведена кодовая таблица (символ – код).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

25. Определение файла, типы файлов. Организация работы с файлами. Использование компонент OpenDialog, SaveDialog

Под файлом в программировании понимается поименованное место на некотором устройстве компьютера куда помещается нужная информация (память, диск, принтер, сканер, …).

Часто, в виде жаргона, эти размещенные в файле данные называют файлами.

В языке Паскаль для организации и последующей работы с файлами (запись данных в файл, чтение из файла) предусмотрен специальный файловый тип переменных.

Операциям над переменными файлового типа соответствуют определенные действия над внешними носителями (дисками, магнитными лентами, принтерами,......).

Переменная типа файл, в языке Паскаль представляет последовательность однотипных компонент, соответствующих последовательности записей на внешнем носителе

Var

Fl1,Fl2:File of <тип компонент>;//типизированные

Lw,Lr:TextFile;//текстовые файлы

f1,f2,f3:File;//нетипизированные файлы

Любая работа с файлом осуществляется через буфер.

Для этого в оперативной памяти выделяется массив из определенного к-ва байтов, который называется буфером.

Когда мы пишем write(fl,x); на самом деле запись идет не в файл, а в этот буфер.

Запись непосредственно в файл происходит в трех случаях:

буфер переполнился, тогда его содержимое переписывается в файл

встретился оператор closefile(fl);

встретился оператор flush(fl);

save open dialogs

Предназначены для выбора требуемого файла на диске через удобное окно просмотра.

Результатом этого выбора является имя и маршрут к файлу помещаемому в переменную строкового типа которые затем используются в процедуре

procedure TForm1.Button1Click(Sender:…);

var ImaFl:string; Fl:file of char;

begin

If OpenDialog1.Execute then

begin

ImaFl:=OpenDialog1.FileName;

AssignFile(Fl, ImaFl );

Reset(Fl);

<работа с файлом Read(Fl, …);>

CloseFile(Fl);

end;

end;

procedure TForm1.Button2Click(Sender:…);

var ImaFl:string; Fl:file of char;

begin

If SaveDialog1. Execute then

begin

ImaFl:=SaveDialog1.FileName;

AssignFile(Fl, ImaFl );

Rewrite(Fl);

<работа с файлом Write(Fl,…);>

CloseFile(Fl);

end;

end;

AssignFile(var F; FileName: string) - связывает файловую переменную F и файл с именем FileName.

Reset(var F[: File; RecSize: word]) - открывает существующий файл. При открытии нетипизированного файла RecSize задает размер элемента файла.

Rewrite(var F[: File; RecSize: word]) - создает и открывает новый файл.

Append(var F: TextFile) - открывает текстовой файл для дописывания текста в конец файла.

Read(F,v1[,v2,…vn]) - чтение значений переменных начиная с текущей позиции для типизированных файлов и строк для текстовых.

Write(F,v1[,v2,…vn]) - запись значений переменных начиная с текущей позиции для типизированных файлов и строк для текстовых.

CloseFile(F) - закрывает ранее открытый файл.

Rename(var F; NewName: string) - переименовывает неоткрытый файл любого типа.

Erase(var F) - удаляет неоткрытый файл любого типа.

Seek(var F; NumRec: Longint) - для нетекстового файла устанавливает указатель на элемент с номером NumRec.

SetTextBuf(var F: TextFile; var Buf[;Size: word]) - для текстового файла устанавливает новый буфер ввода-вывода объема Size.

Flush(var F: TextFile) - немедленная запись в файл содержимого буфера ввода-вывода.

Truncate(var F) - урезает файл, начиная с текущей позиции.

IoResult: integer - код результата последней операции ввода-вывода.

FilePos(var F): longint - для нетекстовых файлов возвращает номер текущей позиции. Отсчет ведется от нуля.

FileSize(var F): longint - для нетекстовых файлов возвращает количество компонентов в файле.

Eoln(var F: TextFile): boolean - возвращает True, если достигнут конец строки.

Eof(var F) ): boolean - возвращает True, если достигнут конец файла.

SeekEoln(var F: TextFile): boolean – возвращает True, если пройден последний значимый символ в строке или файле, отличный от пробела или знака табуляции.

SeekEof(var F: TextFile): boolean - то же, что и SeekEoln, но для всего файла.

BlockRead(var F: File; var Buf; Count: word[; Result: word]) , BlockWrite(var F: File; var Buf; Count: word[; Result: word]) - соответственно процедуры чтения и записи переменной Buf с количеством Count блоков.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

26. Понятие объекта и класса. Объясните на примерах, что такое инкапсуляция наследственность и полиморфизм, раннее и позднее связывание

Основным понятием ООП является объект, который в Delphi представляет собой переменную нового структурированного типа, описываемого с помощью ключевого слова Class. (Объекты всегда распределяются в куче, переменная типа класс это указатель)

Переменная типа Class под одним именем объединяет данные различных типов (поля) и процедуры их обработки (методы)

Объединение данных и процедур их обработки, называемое инкапсуляцией, с приданием ему свойств наследования и полиморфизма оказалось удивительно плодотворным

Любой класс может быть порожден от другого класса.

Класс Tobp порожден от класса Tobr.

Tobr – «класс родитель», а Tobp – «класс-потомок».

Порожденный класс (Tobp) автоматически наследует поля и методы своего родителя (в приведенном примере a, b, F1) и обогащает их новыми (c, d, e, P1).

Принцип наследования позволяет эффективно использовать уже наработанный задел программ и на его основе создавать классы для решения все более сложных задач. Это приводит к тому, что программный продукт представляет собой ветвящееся дерево классов.

Прародителем всех классов в Delphi является класс TObject. Все компоненты Delphi представляют собой созданные разработчиками классы.

Свойство полиморфизма позволяет использовать одинаковое название метода для решения сходных но несколько отличающихся у разных родственных классов задач.

Например, метод Add (добавить) имеется у многих компонент Delphi, хотя в каждой компоненте реализуется по разному.

Обеспечивается это тем, что в классе-потомке метод переписываются по новому алгоритму, т.е. перекрывается. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода с разными алгоритмами.

Obp1.Add; Obr1.Add; действуют по разному

Это и называется полиморфизмом объектов.

Полиморфизм можно организовать по-разному: используя раннее связывание метода с полями объекта, и позднее связывание.

Статические методы: характеризуются тем, что связывание метода с полями объекта осуществляется во время компиляции (раннее связывание). Предыдущие листинги иллюстрируют статические методы.

Виртуальные и динамические методы: связываются с объектом во время выполнения программы (позднее связывание).

Если метод объявлен виртуальным или динамическим, то нельзя менять типы и число параметров.

Позднее связывание методов позволяет реализовать казалось бы удивительную способность родительского метода использовать методы своих потомков.

Замещаемый одноименный метод родителя объявляется как динамический или виртуальный с помощью ключевых слов

Procedure Ris;dynamic; (или virtual)

В потомке замещающий метод объявляется директивой

Procedure Ris; override;

Вызов перекрытого метода родительского класса в одноименном методе потомка достигается с помощью зарезервированного слова

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

27. Что такое виртуальный и динамический способы реализации полиморфизма. В чем их различие. Понятие абстрактного метода. Как реализуется вызов родительского метода в методе потомке?

Виртуальные и динамические методы: связываются с объектом во время выполнения программы (позднее связывание).

Если метод объявлен виртуальным или динамическим, то нельзя менять типы и число параметров.

Позднее связывание методов позволяет реализовать казалось бы удивительную способность родительского метода использовать методы своих потомков.

Встретив объявления dynamic или virtual, компилятор создает таблицы соответствия DMT и VMT.

В этих таблицах помещаются адреса точек входа методов.

Таблица VMT “своего” класса хранится в каждом экземпляре объекта в особом, скрытом от программиста поле.

Таблица DMT хранится в VMT.

При каждом обращении к методу компилятор вставляет в соответствующую таблицу код, позволяющий извлечь затем из нее адрес точки входа в подпрограмму

Отличие таблиц DMT и VMT в том, что

DMT содержит адреса только тех методов, которые объявлены как dynamic в данном классе

VMT содержит адреса всех виртуальных методов данного класса: как нововведенных, так и унаследованных от родителей.

Любой класс имеет VMT, т.к. в Delphi все классы являются потомками класса Tobject, который в свою очередь содержит виртуальные методы.

Более того VMT родителя повторяется в VMT наследника с добавлением в нее сведений о виртуальных методах наследника, если таковые имеются

Использование таблиц VMT обеспечивает более быстрый поиск требуемого метода, но они и занимают много памяти при больших проектах. При использовании динамических методов таблицы DMT создаются только для тех классов, которые их имеют, т.е. они экономят память.

Var ColrBack:Tcolor;

Type

Tviz=class(Tobject)

Canvas:Tcanvas;

ColrLine:Tcolor;

x,y,r:word;

Procedure Ris;virtual;abstract;

Procedure Draw(bl:boolean);

procedure Show;

procedure Hide;

procedure MovTo(dx,dy,dr:integer);

end;

Метод Ris в родительском классе ничего не делает

Объекты абстрактного класса не создаются!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

28. Назначение конструктора и деструктора. Приведите примеры их описаний. Их вид в Delphi

Как всякая динамическая переменная, объект перед началом работы с ним должен быть создан конструктором класса TObject Create:

<Имя-переменной-типа-класс> := <тип-класса> . Create;

После окончания работы с объектом, выделенную под него память необходимо освободить деструктором класса TObject Destroy или Free:

<Имя-переменной-типа-класс> . Free;

в состав любого пользовательского класса могут быть введены свои методы Сonstructor и Destructor с дополнительными к Create и Free функциями.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

29. Понятие свойств класса, для чего они нужны? Как они описываются? Приведите примеры классов со свойствами

Правила хорошего тона при ООП требуют: обращение к полям должно осуществляться посредством методов.

Но каждый раз писать имя метода при обращении к полю не всегда удобно, поэтому придумали специальный механизм, регулирующий доступ к полям.

Свойство – это специальный механизм класса, регулирующий доступ к полям.

Свойства объявляются с помощью ключевых слов property… read…write…;

var p:Tip;

property сp:Tip read методr write методw.

Свойство связано с некоторым полем и указывает:

методr – метод чтения содержимого поля

(например, вывод на печать)

методw - метод записи в это поле.

Пример описания свойства

Type

MyСlass=class( )

p:Integer;

//получить значение p

Function Getpole:Tpole;

Begin

Getpole:=p; //отразить p на форме

label1.caption:=inttostr(p);

end;

// присвоить полю значение

Procedure Setpole(p0:Tpole);

Begin

p:=p0

end; //cp – имя свойства

Property cp:Integer read getpole write Setpole;

(Property cp:Integer read p;)

(Property cp:Integer write Setpole;)

...

end;// MyСlass=class

Работа со свойством

Var

Myobj:Myclass;xt:integer;

begin

....

Myobj:= Myclass.Create;

...

Myobj.cp:=10;

Myobj.Setpole(10);//это эквивалентно

Myobj.p:=10; // так можно но не рекомендуется

....

xt:=Myobj.cp;

xt:=Myobj.Getpole;//это эквивалентно

xt:=Myobj.p; //так можно но не рекомендуется

1. Понятия алгоритм, программа, процесс. Дайте формальное определение алгоритма и поясните его свойства. Понятия примитива и языка программирования.(2)

Алгоритм – это последовательность действий, в результате выполнения которых получается решение поставленной задачи.

Программа является записью алгоритма разработанного для решения некоторой задачи на компьютере.

Процесс – это деятельность связанная с выполнением программы, что тоже самое с выполнением компьютерного алгоритма. Например, в современных компьютерах можно запустить несколько параллельных, взаимодействующих между собой процессов.

Эта проблема решается путем создания четко определенного набора составных блоков (кубиков), из которых могут конструироваться описания алгоритмов. Такие блоки называются примитивами.

То, что примитивам даются точные определения, устраняет многие проблемы неоднозначности и одновременно требуют одинакового уровня детализации для всех описываемых с их помощью алгоритмов.

Язык программирования это набор примитивов вместе с набором правил, устанавливающих, как эти примитивы могут комбинироваться для представления более сложных идей.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Способы описания алгоритма: синтаксис, семантика, нотации Бекуса, схемы Насе-Шнейдермана, что такое псевдокод (4)

• Каждый примитив состоит из двух частей:

• Синтаксической и семантической.

• Синтаксис – это символьное представление примитива.

• Семантика – это значение (содержание) примитива.

Синтаксис – это символьное представление примитива.

Семантика – это значение (содержание) примитива

Схемы Насе - Шнейдермана позволяют уменьшить громоздкость

Slide 20-21, pres. 32-33

Это обычно набор команд некоторого языка с ослабленными правилами.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Понятия теории составления алгоритмов и решения задач. Четыре этапа решения задач. Методы: аналогий, декомпозиции, стратегии «разделяй и властвуй», «динамического программирования». (30)

Создать алгоритм – это значит найти метод решения задачи. Поэтому, чтобы понять, как создаются алгоритмы необходимо понять процесс решения задач.

Теория решения задач – не является специфической только для компьютерных наук

Понять существо задачи; Вникнуть в самую суть. Попытаться сходу ее решить не начиная изучение литературы.

Разработать план решения задачи. (Предложить идею алгоритма); На этом этапе появляется активное отношение к изучению литературы, что более плодотворно, читаешь не впрок, а с определенной целью.

Выполнить план. (Сформулировать алгоритм и написать программу). Возможно, для начала получить грубое решение, не учитывающее всяких нюансов.

Оценить точность решения

Попробовать простейшие, предельные случаи. Выяснить ограничения. Анализ полученных результатов – тестирование

Чаще всего выручает поиск решения по аналогии с уже известным решением похожих; Поэтому надо много читать, как решаются те или иные задачи. Т.е. учиться.

Если задача оказалась оригинальной, тогда используем - Поэтапное уточнение: разбить задачу на ряд подзадач, которые в свою очередь разбиваются на более простые. Нисходящий метод (индукция): стратегии разделяй и властвуй, динамическое программирование.

Иногда надо просто взглянуть на задачу «трезво», или «с другой стороны» с общих позиций (дедукция).

crv

Задача разбивается на независимые подзадачи (части) меньшей сложности (размерности).

Каждая подзадача решается отдельно.

Из отдельных решений подзадач строится решение исходной задачи используя рекурсивный алгоритм.

Эффективность алгоритма при этом может в некоторых случаях повышаться, если на каждом шаге разбиение производится на части приблизительно одинаковой размерности.

Не всегда удается разбить задачу на независимые подзадачи.

Если же подзадачи оказываются зависимы, т.е. имеют общие под-подзадачи, то метод "разделяй и властвуй" делает лишнюю работу, решая некоторые подзадачи по нескольку раз, увеличивая тем самым трудоемкость алгоритма.

Рекурсивный алгоритм в этом случае будет неэффективен и лучше использовать итерационный

Sdp

задача погружается в семейство задач той же природы (другими словами, разбивается на зависимые (могут пересекаться) подзадачи);

каждая подзадача решается отдельно один раз и необходимые значения решений подзадач запоминаются, что позволяет не решать снова уже встречавшиеся ранее подзадачи;

для исходной задачи строится возвратное рекуррентное соотношение, связывающее между собой необходимые значения решений зависимых подзадач

Программируется итерационный алгоритм.

Стратегия метода динамического программирования это попытка свести рассматриваемую задачу к более простым, однако зависимым задачам и опираясь на полученные и сохраненные в массиве результаты решения более простых, решить и исходную задачу.

Эта стратегия за счет увеличения затрат памяти позволяет избавиться от повторного решения пересекающихся подзадач.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Понятие универсального алгоритмического языка. Приведите пример простейшего универсального языка программирования. Как на нем реализовать оператор присваивания y:=x не уничтожая x .(5)

Мы уже знаем, что основой всех языков являются машинные языки. Все остальное это надстройки, позволяющие облегчить программирование тех или иных задач

Построим простейший язык, который будет содержать необходимый набор данных и операторов

Современные языки высокого уровня оперируют с разнообразными структурами данных: числа, массивы, строки, ......, объекты.

Однако после всех трансляций, процессор манипулирует битовыми комбинациями (01).

Поэтому наш простейший язык манипулирует только данными типа: комбинация двоичных разрядов произвольной длины:

{01101010000101010000010}

имя переменной: обозначает такую комбинацию

a,cd,ab,e,….

3 оператора присваивания:

cleare <имя>; очистка строки

incr <имя>; увеличить на 1 (101®110)

decr <имя>; уменьшить на 1

(если перемен. = 0, то

дальнейшего изменения

не будет)

Один оператор цикла:

While <имя> not 0 do

<повторяемые операторы>

...

...

end;

Это знакомый нам оператор While с одним условием имя ? 0.

Присвоить переменной х значение 3 (011):

cleare x;

incr x, incr x, incr x;

Присвоить z значение х (z:=x):

clear z;

While x not 0 do;

Incr z;

decr x;

end;

при этом однако уничтожается х. Это можно легко исправить, введя промежуточную переменную y .

Присвоить z значение х не уничтожая х:

clear y; clear z;

While x not 0 do; присвоить у значение х

Incr y;

decr x;

end;

While y not 0 do; присвоить z и x значение y

Incr z;

Incr x;

decr y;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5. Понятие машины Тьюринга. Нарисуйте и объясните ее работу. Как записывается команда и программа. Роль МТ в теории компьютерных наук. Тезис Черча, теорема Геделя.(6)

Блок управления считывает и записывает символы на ленте с помощью головки.

Лента бесконечна в обоих направлениях, разделена на ячейки, в каждую может быть помещен символ из определенного набора (алфавита внешних символов).

В любой момент вычислений МТ находится в одном из возможных состояний (их количество конечно).

Имеется алфавит внешних символов (a1...al)

Имеется алфавит состояний {q1...qk}. Имеется состояние начало вычислений, состояние останов.

Вычисления МТ состоит из последовательности шагов (тактов), выполняемых блоком управления.

Команда МТ(qi aj ® qmamR) считывается символ aj находясь в состоянии qi из ячейки

запись символа am в ячейку

R – сдвиг влево, вправо, оставить на месте

изменение состояния qm®qn.

Программа представляет последовательность таких команд сообщающих что делать исходя из состояния и содержимого считываемой ячейки aj .

Будучи по своей природе абстрактной, МТ может быть воплощена в разнообразных формах.

Фактически, все современные универсальные ВМ являются машинами Тьюринга, за исключением конечности их памяти.

Подобное сходство совсем не случайно, ведь МТ представляет саму суть вычислительного процесса.

Поразительным результатом математических исследований (Гедель) является обнаружение существования такого класса функций, для которых не существует алгоритмического вычисления. Такие функции получили название невычислимых.

Понимание границы между вычислимыми и не вычислимыми функциями равнозначно пониманию ограничений свойственным любым компьютерам в принципе.

Если функция вычислима то ее можно вычислить с помощью некоторой МТ и наоборот. Если удается доказать, что с помощью МТ ее нельзя вычислить, то она невычислима в принципе.

Значение МТ для современной теории вычислительных систем состоит в предположении, что все вычислимые функции могут быть описаны в виде программы для некоторой МТ

Это является содержанием Тезиса Черча-Тьюринга.

Всегда ли принципиально разрешимая задача имеет практическое решение?

Под размерностью задачи обработки данных понимают число исходных данных в ее формальном описании (далее n). Трудоемкость растет с увеличением количества данных.

Временной сложностью алгоритма называется время, затрачиваемое алгоритмом как функция размерности задачи, которую он решает Tn=f(n)

Емкостной сложностью алгоритма называется объем памяти, требуемый для его реализации как функция размерности Бn=f(n)

Задача(алгоритм) считается тем более сложной, чем больше время тратит на ее расчет ВМ

Так задача линейного поиска считается более сложной O(n/2) чем поиск двоичный O(log2n)

Алгоритм сортировки пузырьком O(n2) более сложен, чем алгоритм слияния O(n?log2n), хотя затраты на программирование наоборот

Задача о рюкзаке O(2n) еще боле сложная чем сортировка. Практически неразрешимая при n>30

Класс полиномиальных задач обозначается P

Класс недетерминированных полиномиальных НП для которых почти доказана полиномиальность (комивояжера, задачи прямого перебора)

Эта гипотеза пока не доказана в виде теоремы, однако имеется много фактов в подтверждении ее справедливости.

Следствие ТЧ-Т: вычислительная мощность МТ выше, чем у любой возможной ВМ.

То есть МТ представляет теоретический предел возможностей ВМ существующих на практике.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6. Понятие рекурсии. Определение рекурсивного алгоритма. Какие задачи решаются рекурсивными методами. Разбиение задачи на подзадачи – элементарные, тривиальные. Как организуются условия окончания рекурсивного алгоритма. Приведите примеры. (7,28)

Рекурсивным называется описание объекта, частично состоящее и определяемое с помощью самого описываемого объекта. После того, как вы познакомитесь с рекурсией, вы обнаружите, что она встречается достаточно часто.

Рекуррентное уравнение называется правильным, если значения аргументов у любой из функций в правой части соотношения меньше значения аргументов у функций в левой части соотношения;

если аргументов несколько, то достаточно уменьшения одного из них.

F(n,x)=F(n-1,x)

U(n,x)=U(n-1,x-2) В общем виде рекурсивный алгоритм P можно выразить как алгоритм, состоящий из комбинации последовательности действий S и самого P:

P = S (a,b,…, P).

На каждом шаге n комбинация

Pn = S (an-1,bn-1,…, Pn-1).

представляет элементарную задачу в предположении, что an-1,bn-1,…, Pn-1 уже вычислены.

Такое определение называется индуктивным

Решать задачу рекурсивно

(т.е. составить рекурсивный алгоритм это значит разложить задачу на подзадачи, которые затем подобным образом (т.е. рекурсивно) разбить на еще меньшие подзадачи;

на определенном уровне подзадачи становятся настолько простыми, что могут быть решены тривиально;

когда решены все элементарные подзадачи, тогда подзадачи составленные из элементарных тоже будут выполнены.

Исходная задача окажется выполненной, когда будут выполнены все подзадачи ее образующие.

Алгоритм разбиения исходной задачи на подзадачи записывается в виде рекуррентного соотношения.

При выполнении рекурсивной подпрограммы вначале осуществляется многократный переход от некоторого текущего уровня (i=n) организации алгоритма к нижнему (или верхнему) уровню последовательно, до тех пор, пока, наконец не будет получено разбиение до тривиального решение задачи.

В вышеприведенных примерах: при i=0 или i=1.

При обратном переходе от полученного тривиального решения к решению для заданного i=n выполняется решение последовательности элементарных подзадач с нарастанием i (или убыванием)

При каждом переходе к следующему уровню (рекурсивной активации процедуры) происходит запоминание всех ее локальных и формальных параметров.

В результате после n-й активации в памяти будет находиться список из n+1 комплектов таких параметров.

Организован этот список таким образом, что комплект параметров, засланный в список последним, будет вызываться первым, и наоборот. Такая структура организации списка данных в памяти получила название стек.

Под организацию стека каждой программе системой Delphi выделяется определенный ограниченный ресурс оперативной памяти.

Подобно любым циклическим алгоритмам рекурсивные подпрограммы могут приводить к своеобразному зацикливанию, когда в процессе последовательного перехода к нижнему уровню не выполняется условие, обеспечивающее нахождение тривиальной подзадачи.

Обычно окончание реализуют след. образом:

Рекурсивный параметр n при каждом обращении уменьшают

n-1 или увеличивают n+1

проверяют условие достижения тривиальной задачи

if (n=Nmin) или if(n=Nmax).

Важно также сделать оценку максимальной глубины рекурсии чтобы убедиться, что она достаточно мала и не приводит к переполнению программного стека

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7. Постановка задачи оптимального выбора. Двоичное дерево решений. Метод полного перебора. Приведите общую процедуру полного перебора вариантов. Метод ветвей и границ –как реализуется проверка по стоимости, эвристические методы. (1,27)

Каждый узел в дереве интерпретируется как результат выполнения некоторой последовательности шагов решения задачи перебора вариантов.

Ветвь в дереве при этом соответствует принятию очередного решения (шагу), которое ведет к более полному решению. В игровых деревьях, например, каждая ветвь соответствует ходу игрока.

Листы (узлы у которых нет выходящих ветвей) представляют собой окончательные варианты решений.

Цель состоит в том, чтобы из всех возможных вариантов решений найти наилучший по заданному критерию путь от корня до листа при выполнении некоторых условий-ограничений.

Форма дерева, критерий оптимальности и условия-ограничения определяются конкретной задачей.

В процедуре Vbr(i,wt,oct) для более компактной реализации суммирования веса и стоимости введены дополнительно два формальных параметра – wt – суммарный вес текущей выборки и oct – общая ее стоимость.

Вначале , т.е. максимальная стоимость

всех элементов.

Затем при движении вдоль выбираемого пути, по мере не включения элементов из oct вычитается их стоимость.

Таким образом oct в каждом узле имеет смысл стоимости, которую еще можно достичь, двигаясь от данного узла до листа.

Критерием

<приемлемости, не включения>

т.е. возможности продолжения построения текущей выборки без данного элемента, будет то, что после данного исключения общая стоимость текущей выборки oct будет не меньше полученной до этого стоимости Cmax оптимальной выборки находящейся в Sopt:

oct-a[i].c>Cmax.

Ведь если выполняется это неравенство, то продолжение поиска, хотя он и может дать некоторое решение, не приведет к более оптимальному решению, чем то, что уже было достигнуто. Следовательно, дальнейший поиск на текущем пути бесполезен.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8. Что такое ключ записи и основное требование к нему. Опишите алгоритм линейного поиска с барьером. Какова его эффективность? (9,24)

Обычно запись содержит некое ключевое слово (ключ), по которому ее находят среди множества других аналогичных записей.

В зависимости от решаемой задачи ключом может, например, служить фамилия или учетный номер или адрес.

Основное требование к ключу в задачах поиска и сортировки состоит в том, чтобы операция проверки на равенство была корректной.

Поэтому, например, в качестве ключа не следует выбирать действительное число, т.к. из-за всегда возможной ошибки округления, поиск нужного ключа может оказаться безрезультатным, хотя этот ключ в массиве имеется.

Для применения более эффективных методов поиска значения ключей должны допускать упорядоченность (т.е. проверку на > или <).

Желательно, чтобы в массиве записей не было повторяющихся ключей.

используется когда нет никакой дополнительной информации о хранении разыскиваемых данных.

Он представляет собой последовательный перебор массива до обнаружения требуемого ключа или до конца, если ключ не обнаружен:

i:=1; n1:=n+1;

While (a[i].k<>x) and (i<n1) do i:=i+1;

if i=n1 then ’элемент не найден’

else ’элемент I’;

Видно, что на каждом шаге требуется увеличивать индекс и вычислять логическое выражение с двумя проверками Так как элемент х с равной вероятностью может быть как вначале, так и в конце массива, то среднее количество повторений в цикле равно n/2.

Ввиду того, что в приведенном алгоритме в каждом цикле две проверки, его эффективность (по к-ву проверок) пропорциональна количеству элементов n, что обозначается О(n).

А можно ли уменьшить здесь затраты на поиск?

Единственная возможность – попытаться упростить логическое выражение, ограничившись одной проверкой в цикле с помощью введения вспомогательного элемента - барьера, который предохраняет от перехода за пределы массива

a[n+1].k=x; i=1;

While a[i].k<>x do i:=i+1;

if i=n+1 then ‘элемент не найден’

else ‘элемент i’;

Эффективность этого алгоритма в два раза выше предыдущего, и оценивается как О(n/2).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9. Что такое ключ записи и основное требование к нему. Опишите алгоритм двоичного поиска и дайте оценку его эффективности. (8,3)

-//-

В этом алгоритме отсутствует внутри цикла проверка совпадения a[m].k=x.

На первый взгляд это кажется странным, однако тестирование показывает, что в среднем, выигрыш от уменьшения количества проверок превосходит потери от нескольких «лишних» вычислений до выполнения условия i=j.

Видно, что алгоритм двоичного поиска рекурсивен, т.е. более сложная задача сводится к последовательности простейших:

Poisk(a[1]..a[n])= if a[m].k<x then Poisk(a[1]..a[n/2])

else Poisk(a[n/2+1]..a[n])

Легко подсчитать, что эффективность этого алгоритма оценивается как

2m=n ? m=log2n

это обозначается как О(lоg2n).

Т.е. для n=128 log2128=7 128/2/7@9

этот метод в 9 раз эффективнее линейного поиска с барьером

для n=1024 – в 50 раз,

для n=106 – в 2500 раз.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10. Изложите алгоритмы трех основных методов прямой сортировки массива и охарактеризуйте их эффективность. (10,23)

Для i=1..n-1 выполняется следующий элементарный алгоритм:

начиная от n до i последовательно проверяем упорядоченность двух соседних элементов a[j].k и a[j-1].k, и если они не упорядочены, то меняем их местами.

В результате такого обмена минимальный элемент перемещается на место i

for i:=1 to n-1 do

for j:=n downto i do

if a[j-1].k>a[j].k then

begin

r:=a[j-1];

a[j-1]:=a[j];

a[j]:=r;

end;

Допустим, что массив a[1..n] разбит на две части

a1...ai-1, ai...an

– в первой части элементы упорядочены, во второй – нет.

При i=2 такое разбиение получается автоматически, т.к. массив из одного элемента тривиально упорядочен.

На каждом шаге i=2..n выполняем элементарный алгоритм:

берем очередной элемент r из неупорядоченной части и включаем его в «нужное» место упорядоченной части.

a1...ai-1, ai...an

Для поиска нужного места в нижеприведенном алгоритме используется барьер a[0].k:=x.

Элементы, начиная от i-1 до нужного j сдвигаются на одну позицию: a[j]:=a[j-1]

for i:=2 to n do

begin

r:=a[i]; //элемент из неупор части

x:=a[i].k;//ключ этого элемента

a[0].k:=x; //барьер

j:=i;

while x<a[j-1].k do //поиск места

begin

a[j]:=a[j-1]; //сдвиг

j:=j-1;

end;

a[j]:=r;//вставка в нужное место

end;

Для i=1..n-1 выполняется следующий элементарный алгоритм: среди элементов ai...an выбираем минимальный am и переставляем местами элементы i-й и m-й.

a1...ai-1, ai…am...an

В результате на первое место станет самый минимальный, на второе – следующий минимальный и т.д

for i=1 to n-1 do

begin

m:=i; //поиск m

for j=i+1 to n do

if a[j].k<a[m].k then m:=j;

r:=a[i]; //перестановка

a[i]:=a[m];

a[m]:=r;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11. Изложите алгоритм метода быстрой сортировки Хоара. (11,22)

Выберем значение ключа среднего m-го элемента x=a[m].k (m можно выбирать либо делением пополам, либо случайным образом, как рекомендует автор).

i=1 j=n

Будем просматривать массив слева до тех пор пока не обнаружим элемент a[i].k>x.

После этого будем просматривать массив справа, пока не обнаружим a[j].k<x.

Поменяем местами элементы a[i] и a[j] и продолжим такой процесс просмотра (слева и справа, обмен) пока оба просмотра не встретятся где-то внутри массива (j<i причем j=i-1).

В результате массив окажется разбитым на

левую часть a[1..j], с ключами меньше (или равными) x

правую a[i..n], i=j+1, с ключами больше (или равными) x.

Var a:mas;n:word;

Procedure QuickSortR;

Procedure sort(L,R: word);

Var i,j:Word; w:Tzp; x:Tk;

begin

i:=L;j=R;

x:=a[(L+R) div 2].k;

Repeat

While a[i].k<x do i:=i+1;

While a[j].k>x do j:=j-1;

if i<=j then begin

w:=a[i];a[i]:=a[j];a[j]:=w;i=i+1;j=j-1;

end;

until i>j;

if L<j then sort(L,j);

if i<R then sort(i,R);

end; //sort

begin

sort(1,n);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12. Изложите алгоритм метода сортировки слиянием массива записей. Его эффективность (12)

Var a:mas; n: word;

Procedure SortSlip;

Procedure SrSL(L,R:word);

var m:word;

begin

if L<>R then begin

m:=(L+R) div 2;//разбиение

SrSL(L,m);

SrSL(m+1,R);

Slip(L,m,R);

end;

end;

begin

SrSL(1,n)

end; var a:mas;

Procedure Slip(L,m,R:word);

Var c: mas; i,j,k:word;

begin

i:=L; k=1; j:=m+1;

while (i<=m) and (j<=R) do

if a[i].k<a[j].k

then begin c[k]:=a[i]; Inc(i); Inc(k) end

else begin c[k]:=a[j]; Inc(j); Inc(k) end;

while i<=m do

begin c[k]:=a[i]; Inc(i); inc(k) end;

while j<=R do

begin c[k]:=a[j]; Inc(j); Inc(k) end;

k=0;

for i:=L to R do

begin Inc(k); a[i]:=c[k] end;

end;

является иллюстрацией того, каким образом за счет использования дополнительных рабочих массивов можно значительно повысить эффективность сортировки.

Идея метода основана на следующем эффективном алгоритме слияния.

Пусть внутри массива a[1..n] имеется два смежных отсортированных участка (например, по возрастанию ключа) a[L, m], a[m+1,R] 1?L<R?n.

a1….aL...am, am+1...aR….an

Необходимо отсортировать весь участок a[L,R].

Процедура сортировки для этого участка организуется специфическим слиянием двух участков в один рабочий массив

C1…CR-L+1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13. Напишите алгоритмы и дайте сравнительную характеристику эффективности методов прямой и быстрой сортировок обменом элементов. Что такое эффективность метода и как она оценивается. (26)

О(n2) I log2n

Эффективность сортировки

Методы внутренней сортировки классифицируются по времени их работы.

Хорошей мерой эффективности может быть

число сравнений ключей - С

число пересылок элементов - Р.

Эти числа являются функциями С(n), Р(n) от числа сортируемых элементов n.

Быстрые (но сложные) алгоритмы сортировки требуют

при n®? порядка n?lgn сравнений т.е. их эффективность O(n?lgn)

Прямые (простые) методы - O(n2).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

14. Что такое список? Как организуется список на основе динамического массива данных. Понятие стека и очереди. Методы добавления и чтения элемента. (13, 17)

В практике программирования довольно часто встречается задача обработки набора однотипных данных

Inf1, Inf2, ...... Infn,

количество которых n меняется в зависимости от ситуации

Их надо разместить в оперативной памяти и по мере надобности с ними работать: добавлять новые данные в список, освобождаться от старых данных, выдавать информацию о количестве данных, находить нужную информацию

Такой набор программисты называют СПИСКОМ

Простейшая форма организации списка – это динамический массив данных. Данные, размещенные в массиве имеют свои номера (индексы) и это придает эффект «видимости» каждому элементу.

Turn-это список с двумя точками входа. С одной стороны последовательности данные добавляются в список (в конец очереди), с другой стороны списка данные удаляются из него (из начала очереди)

Таким образом реализуется принцип – «первым пришел – первым вышел»

Наглядный пример – трубка, заполняемая шариками, имеющими диаметр, равный диаметру трубки. С одной стороны трубки шарики добавляются в нее, с другой вынимаются.

Stack- это список с одной точкой входа.

Данные добавляются в список и удаляются из него только с одной стороны последовательности (вершины стека)

Таким образом реализуется принцип – «последний пришел – первым вышел»

procedure Tlist.Addk(c:Tinf);

begin

GetMem(a1,(n+1)\* mt);

a1[n+1]:=с; //добавление элемента

if n>0 then begin

for i:=1 to n do // копирование

a1[i]:=a[i];

FreeMem(a,mt\*n);//освобождение

end;

a:=a1; n:=n+1; //теперь имя опять a

end;

procedure Tlist.Read1(var c:Tinf);

begin

if n>0 then begin

c:=a[1]; n:=n-1;

if n>0 then begin

GetMem(a1,n\* mt);

//копирование

for i:=1 to n do a1[i]:=a[i+1];

end else a1:=nil;

FreeMem(a,mt\*(n+1)); //освобождение

a:=a1; //теперь имя опять a

end else «список пуст!!!»;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15. Что такое список? Как организуется однонаправленный список на основе рекурсивного типа данных. Понятие стека и очереди. Методы добавления и чтения элемента. Что такое однонаправленный список с меткой? Как организуется циклический и двунаправленный списки. (14,20))

Здесь используется единственное исключение языка Паскаль – тип (sel) используется раньше, чем он описан

Как видим, описание типа Tsel содержит внутри обращение к самому себе (A:Tsel), т.е. оно рекурсивно.

При этом А является указателем на ячейку памяти точно такой же структуры.

В переменной Inf размещаются данные о некотором элементе списка, причем по крайней мере в одном из полей Inf, а иногда и в отдельном поле записи расположены сведения по которым производится поиск требуемой информации.

Это поле будем обозначать key:Tkey, а сведения называть ключом.

Программисту известен только один указатель sp1, который назовем вершиной стека

Доступ к ячейкам стека возможен только через этот указатель, в котором расположен адрес первой ячейки.

Программисту известны два указателя sp1, spk

Доступ к ячейкам стека возможен только через эти указатели, в которых расположены адрес первой и адрес последней ячеек

В конец очереди через spk добавляют новые ячейки, из начала очереди через sp1 удаляют ячейки.

Методы работы с очередью немного отличаются от методов работы со стеком – при работе с очередью необходимо поддерживать вторую точку входа spk

Поэтому более удобно написать два класса – один для работы со стеком, другой с очередью

Как мы видели, в однонаправленном списке довольно сложно взять элемент изнутри списка

Например последний элемент из очереди просто так удалить не получается, хотя имеется его адрес spk

Так же проблематично вставить элемент внутрь очереди перед заданным

Хотя довольно эффективно работают процедуры удаления и вставки элемента после элемента с заданным адресом ReadAftter, AddAftter.

Однако при их регулярном использовании имеется проблема первого элемента – его надо обрабатывать другой процедурой, а значит каждый раз требуется лишняя проверка

Чтобы избавиться от нее, часто используют Список с меткой

В таком списке от первой ячейки используется только ее адресная часть sp1.A,

в информационной же части могут быть при необходимости размещены данные об этом списке, при этом метка создается методом Add1

Stekm:=create; Stecm.add1(Infm); .

При работе со стеком sp1 остается неизменным

После создания списка с меткой работа с ним производится с помощью единообразных процедур ReadAfter, AddAfter, PoiskAfter.

Циклические списки используются, когда нужно обходить набор элементов в бесконечном цикле.

Они так же позволяют получить доступ к списку начиная с любой позиции, что придает списку некоторую симметрию.

Программа может работать со всеми элементами одинаково без использования меток с помощью процедур RedAfter, AddAfter, PoiskAfter.

организуются когда требуется просматривать список как в одном так и в обратном направлениях.

Мы видели, что в однонаправленном списке довольно просто удалить (вставить) новую ячейку после заданной, но довольно сложно удалить саму заданную ячейку или предыдущую к ней.

Эта проблема легко решается, если ввести рекурсивный тип с двумя адресными ячейками

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

16. Для чего нужны древовидные структуры данных? Приведите и проиллюстрируйте основные определения: порядок узла, степень дерева, глубина, сбалансированность. Напишите класс и методы обхода дерева и его удаления. (15, 25)

При решении игровых задач мы встретились с понятием дерева, которое использовалось в качестве математической модели для представления множества возможных вариантов.

Познакомились с рекурсивным алгоритмом обхода всех узлов и листьев дерева и построенным на нем методом ветвей и границ.

Древовидная модель оказывается довольно эффективной и для представления динамических данных с целью быстрого поиска информации.

Если два узла соединены направленной дугой (x®y) то

х - предшественник (родитель),

у – преемник (сын, дочь).

Корень - узел, у которого нет родителя.

Лист - Узел, не имеющий преемников

Внутренний узел – это узел, не являющийся листом или корнем.

Порядок узла – количество его дочерних узлов.

Степень дерева – максимальный порядок его узлов

Глубина узла равна числу его предков плюс 1.

Глубина дерева – это наибольшая глубина его узлов.

Дерево, представленное выше имеет степень 3 (троичное), глубину 4, корневой узел а, листы d, e, g, k,m

Procedure DeleteTree(var p:Ttree);

Begin

If (p<>Nil) then begin

DeleteTree(p^.A1);

DeleteTree(p^.A2);

. . . . .

DeleteTree(p^.Am);

Dispose(p);

p:=Nil;

end;

Ввиду своеобразной организации эффективность поиска информации в такой динамической структуре данных сравнима с эффективностью двоичного поиска в массиве, т.е. имеет оценку эффективности О(log2n).

Заметим что двоичный поиск в линейном связанном списке организовать невозможно, а эффективность линейного поиска имеет порядок О(n/2).

Конечно, оценка О(log2n) справедлива для сбалансированного дерева, т.е. такого у которого узлы, имеющие только одну дочь, имеют глубину равную или на единицу меньшую глубины дерева.

Begin

Del(proot); inherited free;

end;

Procedure TtreeB.Wrt1;

Procedure Wr(p:Ttree);

begin

if p<>nil then

begin

Wr(p^.A1);

Print(p^.Inf);

Wr(p^.A2);

end;

end;

begin

p:=proot; wr(p)

end;

Function TtreeB.poisk(k:Tkey):Tinf;

begin p:=Proot;

While(p<>nil) and (p^.Inf.key<>k) do

If k<p^.Inf.key then p:=p^.A1

else p:=p^.A2;

if p<>Nil then Result:=p^.inf

else Result.key:=nok;

end;

Обращение к методу

Inf:=tr.poisk(k);

If inf.key=nok then print(’нет ключа’)

else . . .

Procedure Ttree.Add(Inf:TInf);

Var bl:Boolean;

begin

New(w); // создание нового листа:

w^.Inf:=Inf;

w^.A1:=Nil;

w^.A2:=Nil;

if proot=Nil then proot:=w

else

begin

p:=proot;

repeat

q:=p; // q - указатель на предыдущий элемент

bl:=Inf.key<p^.Inf.key;

if bl then p:=p^.A1

else p:=p^.A2;

until p=Nil; // добавление листа

if bl then q^.A1:=w

else q^.A2:=w;

end;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

17. Что такое дерево поиска? Приведите пример идеально сбалансированного и плохо сбалансированного дерева поиска. Процедуры обхода и нахождения элемента в дереве поиска. (16,19)

ключи расположены таким образом, что для любого узла значение ключа у родителя больше чем у левого преемника, при этом значение ключа у левого преемника меньше чем у правого.

Ввиду своеобразной организации эффективность поиска информации в такой динамической структуре данных сравнима с эффективностью двоичного поиска в массиве, т.е. имеет оценку эффективности О(log2n).

Заметим что двоичный поиск в линейном связанном списке организовать невозможно, а эффективность линейного поиска имеет порядок О(n/2).

Конечно, оценка О(log2n) справедлива для сбалансированного дерева, т.е. такого у которого узлы, имеющие только одну дочь, имеют глубину равную или на единицу меньшую глубины дерева.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

18. Что такое хеширование? Понятие хеш-функции, хеш-таблицы, алгоритм разрешения конфликтов. Напишите класс для хеширования. Опишите три способа хеширования, которые вы изучали. (21)

Хеширование (hashing-перемешивание) состоит в том, что ключи данных записываются в так называемую хеш-таблицу (массив).

Идея в том, что при помощи некой простой функции i=h(key) алгоритм хеширования определяет положение искомого элемента в таблице по значению его ключа.

Допустим, имеется набор n записей с ключами в диапазоне 0?key?K, n<K.

Создадим массив (хеш-таблицу):

H:array[0..K] of <тип записей>;

Вначале его очистим H[i]:=0 и наши n записей помещаем в этот массив в соответствии со значением ключа i=key. Затем, используя операторы

zp:=H[key]; //извлекаем из массива

H[key]:=zp; //записываем в массив

Функция h(key) называется функцией расстановки или хеш-функцией.

Ввиду того, что число возможных значений ключа K обычно значительно превосходит возможное количество записей K>>n, любая функция расстановки может для нескольких значений ключа давать одинаковое значение позиции i в таблице.

Например, если M=10 то для key=1, key=11 и key=101 позиция i=h(key)=1.

В этом случае h(key) только определяет позицию, начиная с которой нужно искать запись с ключом key Ввиду такой неоднозначности схема хеширования должна включать алгоритм разрешения конфликтов, определяющий порядок действий, если позиция i=h(key) оказывается занятой записью с другим ключом.

Имеется множество схем хеширования различающихся как выбором удачной функции h(key) так и алгоритма разрешения конфликтов.

Эффективность решения реальной практической задачи будет существенно зависеть от выбираемой стратегии разрешения конфликтов.

содержит следующие действия:

сначала ключ key очередной записи отображается на позицию i=h(key) таблицы

если позиция свободна то в нее размещается элемент с ключом key,

если занята то отрабатывается алгоритм разрешения конфликтов, который находит место для размещения данного элемента.

Один из самых изящных способов разрешения конфликтов состоит в том, чтобы сохранять записи, отображаемые на одну позицию в связанные стеки. Для этого вводится массив из М указателей на связанные стеки

Можно и на массив бинарных деревьев поиска

Обычно выбирают М - простое число, немного большее чем предполагаемое количество записей.

Хеш-функция выбирается в виде i:=Key mod M.

Работу с таблицей удобно организовать посредством класса ТН

Преимущество рассмотренного метода заключается в том, что связанные хэш-таблицы никогда не переполняются, довольно просто осуществляется вставка, удаление и поиск элементов.

Недостаток таких таблиц в том, что если данные недостаточно равномерно перемешаны по ключу, то некоторые списки могут оказаться очень длинными в то время как большинство других будут пустыми, при этом поиск будет замедляться.

Для избавления от этого недостатка нужно придумать другую функцию хеширования.

Иногда используют следующую функцию

i:=(Key+P) Mod M,

где Р - простое число.

Можно Р генерировать с помощью датчика случайных чисел, при этом вначале записи и поиска датчик устанавливается в одинаковое начальное положение

Перемешанная таблица Этод метод послужил поводом, для названия (хеширование).

Используется простой массив H[i], функция хеширования i=Key mod M.

Алгоритм разрешения конфликтов следующий:

Если очередная позиция i занята, тогда ищется первая свободная позиция и туда помещается элемент. Для уменьшения эффекта кластеризации можно использовать функцию хеширования с датчиком случайных чисел, описанную выше.

Метод блоков Используется массив одномерных массивов (блоков)

Type Tblok=array[1..P] of Tinf;

Var H:array[0..M-1] of Tblok;

H1,H2:array[0..M-1] of Tblok;

блоки переполнения

Размещаются элементы следующим образом:

Вначале обычная функция i:=Key mod M находит номер блока и размещает в H[i] элемент (в конце блока).

Если i-й блок переполнен, то элемент помещается в очередной блок переполнения H1[i].

Этот способ обычно используется при хранении данных на файле. Запись и чтение в (из) файла осуществляется поблочно. Это позволяет значительно ускорить поиск, т.к. чтение из файла по одной записи медленнее, чем сразу блоком.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

19. Работа с разреженными матрицами на основе массива из стеков. Напишите класс, включающий конструктор и методы добавления и чтения элемента в разреженную матрицу (18)

Матрицы, содержащие большое число нулевых элементов называются разреженными

Решение многих прикладных задач (методы теории графов, численные методы решения дифференциальных уравнений методом конечных элементов, методы обработки изображений, методы криптографии) связано с обработкой матриц большой размерности n имеющих малое число ненулевых элементов

Использование специальных способов размещения в памяти разреженных матриц и разработка специальных алгоритмов работы с ними позволяет во многих случаях существенно снизить затраты компьютерных ресурсов и уменьшить время вычислений.

Эффективность работы с таким массивом определяется введением класса с продуманным набором методов и создание на этой основе эффективных процедур работы с матрицами

constructor TListms.create(m0,n0:word);

Var i:word;

begin m:=m0; n:=n0;

inherited create;

GetMem(ms,4\*m);

for i:=1 to m do begin

ms[i]:=Tlists.create;

ms[i].sp1:=nil

end;

end;

Procedure TListms.add(i,j:word;a:extended);

Var inf:Tinf;

begin

inf.a:=a; inf.j:=j;

ms[i].adds(inf);

end;

Function TListms.read(i,j):extended;

Var inf:Tinf;

begin

result:=ms[i].readsj(j);

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20. Понятия статического и динамического распределения памяти. Что такое указатель? Операции над указателями. Что такое динамическая переменная? Наложение переменных с помощью указателя

Все команды и данные программы во время ее выполнения размещаются в определенных ячейках оперативной памяти

При этом часть данных размещается в ячейки памяти, отводимой под программу еще на этапе компиляции и в процессе работы программы их адреса относительно начала программы не изменяются.

Такое размещение данных и команд называется статическим

Соответствующие этим данным переменные называются статическими переменными Возможна также организация динамического размещения данных

В этом случае под некоторые данные и программы память выделяется непосредственно во время выполнения по мере надобности, а после решения требуемой задачи память освобождается для других данных

Соответствующие таким данным переменные называются динамическими переменными

Для организации динамического распределения памяти используются переменные специального типа – указатели, которые обеспечивают работу непосредственно с адресами ячеек памяти.

Под каждую переменную типа указатель отводится ячейка объемом 4 байта, в которой помещается адрес нужной переменной Type Pk=^тип;

Pmi=^array[1..20] of integer;

Pb=^byte;

Var u,v:тип;

p,q:pointer;

a,b:Pk;

mi:Pmi

k:Pb;

s:^char;

Здесь р,q – нетипизированные указатели;

а,b,mi,k,s – типизированные указатели, т.е. они указывают, что, например, в ячейках начиная с адреса а размещается переменная указанного типа <тип>

Адрес статической переменной указателю можно присвоить следующим образом Если v, u – обычные статические переменные, то их адреса (адреса первого байта) можно получить с помощью специальной функции

p:=Addr(u); a:=Addr(v);

Очистка адреса из указателя осуществляется с помощью специальной функции Nil:

p:=Nil; a:=Nil,

довольно часто используется проверка условия

if p<>Nil then ...

if a=Nil then ... Значениями указателей являются адреса переменных, размещенных в памяти.

Значение одного указателя можно передать другому с помощью оператора присваивания, например:

p:=q;

a:=b;

При этом надо помнить, что в операторе присваивания типизированные указатели должны быть одного типа.

Используя нетипизированные указатели, можно передать адрес между указателями разного типа, например так:

p:=i; k:=p; Использование указателей позволяет «накладывать» переменные разных типов друг на друга, интерпретируя по-разному данные, расположенные по некоторому адресу

В этом случае мы имеем возможность неявного преобразования типов

Var Ch:Char;

i:^byte;

begin

i:=Addr(ch);

ch:=’A’;

Write(‘код ’,ch,’ = ’,i^:4);

Будет выведено сообщение

код А = 160

Процедура

New(a:<типизированный указатель>);

находит в куче свободный участок памяти, размер которого позволяет разместить тип данных а и присваивает указателю а значение адреса первого байта этого участка.

После этого данный участок памяти закрепляется за программой и с ним можно работать через возникшую в программе переменную a^.

Такие переменные называются динамическими.

После того как необходимость работы с этой переменной отпала, данный участок памяти освобождается с помощью процедуры

Dispose(a);

При этом адрес остается в указателе a

При работе как с типизированными так и с нетипизированными указателями аналогичные New, Dispose действия выполняют процедуры

GetMem(P:pointer;size:Word);

FreeMem(P,size);

здесь size - количество байт выделяемой памяти начиная с адреса, помещаемого в указатель Р.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

21. Понятие подпрограммы, для чего они нужны? Как описывается подпрограмма в виде Procedure? В виде Function? Что такое формальные и фактические параметры? В чём различие параметров-значений и параметров-переменных? Как вызвать процедуру? Функцию? Как оформить набор подпрограмм в виде библиотечного модуля (Unit)?

Подпрограмма – это последовательность операторов, оформленная таким образом, что ее можно вызвать по имени из любой программы, которую вы составляете, передать ей определенные параметры и получив требуемый результат продолжить вычисления. Вызов процедуры

…….

имяП(<список фактических параметров>);

Imap(x,y,m,k,j,S);

Вызов функции

…….

Y:=имяФ(<список фактических параметров>);

s:=y\*имяФ(сп.факт.пар.1)+y/имяФ(сп.факт.пар.2);

Z:=Imaf(a,b,m,k,j,S);

Y должно иметь типФ

Для каждого параметра – значения транслятор внутри подпрограммы резервирует дополнительные ячейки памяти в соответствии с типом параметра.

При вызове подпрограммы, происходит пересылка фактического параметра в эти ячейки памяти, после чего выполняется подпрограмма.

Значение ячейки, где находился сам фактический параметр не изменяется.

Этот механизм обеспечивает защищенность фактического параметра.

Фактическим параметром может быть константа, переменная или арифметическое выражение

Переменная

Фактическим параметром может быть только имя переменной.

При вызове подпрограмм передается адресс ячейки переменной, в которой находится фактический параметр, и все действия производятся над этой ячейкой.

После работы подпрограммы в ячейке фактического параметра при необходимости будет находиться результат.

Набор подпрограмм которые могут быть использованы при разработке целого ряда программ удобно оформить в виде отдельных тематических библиотек.

Для организации таких библиотек в Паскале введены модули.

Модуль (Unit) – это автономно компилируемая программная единица.

Модуль состоит из интерфейсной части, доступной для других программ, и исполняемой части которая скрыта.

В интерфейсной части располагаются необходимые константы, переменные и все заголовки подпрограмм помещенных в данную библиотеку.

В исполняемой части располагаются описания констант, переменных и подпрограмм.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

22.Как выводить графики в Delphi, используя компоненту Tchart? Опишите последовательность действий по настройке Tchart и фрагмент программы вывода графика y=sinx.

1) на закладке 3D Отключаем ^ 3Dimention

2) на закладке Panel

Выбираем Panel Color

3) на закладке Titles

Пишем Мой график

4) Нажимаем Add

Добавляем новый график

После нажатия Add

Вам предлагается выбрать вид графика.

Выбираем Line -

график в виде ломанной линии соедaиняющей последовательность точек с координатами (Xi,Yi)

После выбора двух графиков нажмем закладку Series

На закладке Border выберем толщину Width и стиль Style линий графика

На закладке Color выберем цвет линии

procedure TForm1.Button1Click(Sender: TObject);

var h,x,y,z:extended;

i:word;

begin

Chart1.SeriesList[0].Clear;

Chart1.SeriesList[1].Clear;

h:=0.1;

x:=0;

for i:=1 to 100 do begin

y:=sin(x);

z:=1/(x+1)

Chart1.SeriesList[0].AddXY(x,y);

Chart1.SeriesList[1].AddXY(x,z);

x:=x+h;

end;

end;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

23. Как выводить графики, используя компоненту TImage Опишите последовательность действий по настройке TImag и фрагмент программы вывода закрашенного квадрата внутри круга

Любое изображение на экране дисплея формируется из набора светящихся точек, получивших название пиксели.

Процесс отображения осуществляется с помощью специальной микросхемы называемой графическим адаптером.

Работа графического адаптера осуществляется под управлением специальной программы - драйвера.

Адаптер содержит порты ввода-вывода информации, оперативную память в которой помещается таблица, содержащая информацию о каждом светящемся на экране пикселе (его координате, цвете, яркости). Рисование картинки в системе Delphi происходит в рамках создаваемого проекта на форме.

Наиболее удобно для этого пользовать компоненты Image (Addition)или Paintbox (Sistem). Можно рисовать непосредстввенно на форме.

Компоненты Image и PaintBox представляют собой выделенный на форме прямоугольник с канвой, на которую можно наносить изображения. Размеры прямоугольника в пикселях определяются на этапе проектирования.

Основное достоинство Image – это возможность записывать и считывать нарисованную картинку в файл.

Класс TCanvas

предоставляет в ваше распоряжение таблицу пикселей с координатами

pxmax:=Image1.Width; pymax:=Image1.Height;

Вместе с набором графических инструментов

Рисование начинается с очистки фона окна:

Image1.Picture:=Nil;

или

Image1.Canvas.Brush.color:=clWhite;

Image1.Canvas.Rectangle(0,0,pxmax,pymax);

Оператор - Установить цвет выбранного пикселя

Image1.Canvas.Pixels[px, py]:=ClBlack;

Можно начинать рисовать по точкам:

With Image1 do begin

dp:=Height div 4;

Px0:=Width div 2;

Py0:=Height div 2;

For px:=px0-dp to px0+dp do

canvas.pixels[px,py0]:=ClRed;

For py:=py0-dp to py0+dp do

canvas.pixels[px0,py]:=ClNavy;

End;

вычертится красно-синий крест. Он будет нарисован довольно тонккими линиями, т.к. рисуется по точкам.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

24. Что такое записи в языке Pascal? Как они описываются, в чем особенность вариантной части? Приведите примеры использования вариантной части.

Запись – это структура данных, в которой под одним именем объединены переменные, в общем случае разного типа, которые называются полями записи

Type

Tzap=record

a,b,c:Typ1; Разделы фиксированной части

e,f:Typ2; записи

........

z,x,y,u:Typn;

case byte of

1:(d:Typ01; Разделы вариантной части

g:Typ02); записи

........

m:(p,q:Typ1m) <-вариант m

end;

Var

Z1,Z2:Tzap;

Возможны записи, имеющие только фиксированную часть (отсутствует вариантная часть) или имеющие только вариантную часть (отсутствует фиксированная часть). Вариантная часть только одна и ставится в конце записи.

Каждому полю фиксированной части отводится ячейка памяти в соответствии с его типом.

Для вариантной части под все варианты полей отводится одна и та же область памяти, объем которой равен максимальному из вариантов полей.

Поэтому соответствующие поля разных вариантов располагаются на одних и тех же байтах памяти!

В вариантной части недопустимы длинные строки (string)!

Type

Tzp3=record

Case byte of

1:(a:array[0..255] of byte);

2:(b:array[0..255] of char);

end;

........

Var z3:Tzp;

begin

With z3 do begin

for i:=0 to 255 do

begin a[i]:=i; Writeln(b[i],a[i]);end;

end;

end;

Будет выведена кодовая таблица (символ – код).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

25. Определение файла, типы файлов. Организация работы с файлами. Использование компонент OpenDialog, SaveDialog

Под файлом в программировании понимается поименованное место на некотором устройстве компьютера куда помещается нужная информация (память, диск, принтер, сканер, …).

Часто, в виде жаргона, эти размещенные в файле данные называют файлами.

В языке Паскаль для организации и последующей работы с файлами (запись данных в файл, чтение из файла) предусмотрен специальный файловый тип переменных.

Операциям над переменными файлового типа соответствуют определенные действия над внешними носителями (дисками, магнитными лентами, принтерами,......).

Переменная типа файл, в языке Паскаль представляет последовательность однотипных компонент, соответствующих последовательности записей на внешнем носителе

Var

Fl1,Fl2:File of <тип компонент>;//типизированные

Lw,Lr:TextFile;//текстовые файлы

f1,f2,f3:File;//нетипизированные файлы

Любая работа с файлом осуществляется через буфер.

Для этого в оперативной памяти выделяется массив из определенного к-ва байтов, который называется буфером.

Когда мы пишем write(fl,x); на самом деле запись идет не в файл, а в этот буфер.

Запись непосредственно в файл происходит в трех случаях:

буфер переполнился, тогда его содержимое переписывается в файл

встретился оператор closefile(fl);

встретился оператор flush(fl);

save open dialogs

Предназначены для выбора требуемого файла на диске через удобное окно просмотра.

Результатом этого выбора является имя и маршрут к файлу помещаемому в переменную строкового типа которые затем используются в процедуре

procedure TForm1.Button1Click(Sender:…);

var ImaFl:string; Fl:file of char;

begin

If OpenDialog1.Execute then

begin

ImaFl:=OpenDialog1.FileName;

AssignFile(Fl, ImaFl );

Reset(Fl);

<работа с файлом Read(Fl, …);>

CloseFile(Fl);

end;

end;

procedure TForm1.Button2Click(Sender:…);

var ImaFl:string; Fl:file of char;

begin

If SaveDialog1. Execute then

begin

ImaFl:=SaveDialog1.FileName;

AssignFile(Fl, ImaFl );

Rewrite(Fl);

<работа с файлом Write(Fl,…);>

CloseFile(Fl);

end;

end;

AssignFile(var F; FileName: string) - связывает файловую переменную F и файл с именем FileName.

Reset(var F[: File; RecSize: word]) - открывает существующий файл. При открытии нетипизированного файла RecSize задает размер элемента файла.

Rewrite(var F[: File; RecSize: word]) - создает и открывает новый файл.

Append(var F: TextFile) - открывает текстовой файл для дописывания текста в конец файла.

Read(F,v1[,v2,…vn]) - чтение значений переменных начиная с текущей позиции для типизированных файлов и строк для текстовых.

Write(F,v1[,v2,…vn]) - запись значений переменных начиная с текущей позиции для типизированных файлов и строк для текстовых.

CloseFile(F) - закрывает ранее открытый файл.

Rename(var F; NewName: string) - переименовывает неоткрытый файл любого типа.

Erase(var F) - удаляет неоткрытый файл любого типа.

Seek(var F; NumRec: Longint) - для нетекстового файла устанавливает указатель на элемент с номером NumRec.

SetTextBuf(var F: TextFile; var Buf[;Size: word]) - для текстового файла устанавливает новый буфер ввода-вывода объема Size.

Flush(var F: TextFile) - немедленная запись в файл содержимого буфера ввода-вывода.

Truncate(var F) - урезает файл, начиная с текущей позиции.

IoResult: integer - код результата последней операции ввода-вывода.

FilePos(var F): longint - для нетекстовых файлов возвращает номер текущей позиции. Отсчет ведется от нуля.

FileSize(var F): longint - для нетекстовых файлов возвращает количество компонентов в файле.

Eoln(var F: TextFile): boolean - возвращает True, если достигнут конец строки.

Eof(var F) ): boolean - возвращает True, если достигнут конец файла.

SeekEoln(var F: TextFile): boolean – возвращает True, если пройден последний значимый символ в строке или файле, отличный от пробела или знака табуляции.

SeekEof(var F: TextFile): boolean - то же, что и SeekEoln, но для всего файла.

BlockRead(var F: File; var Buf; Count: word[; Result: word]) , BlockWrite(var F: File; var Buf; Count: word[; Result: word]) - соответственно процедуры чтения и записи переменной Buf с количеством Count блоков.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

26. Понятие объекта и класса. Объясните на примерах, что такое инкапсуляция наследственность и полиморфизм, раннее и позднее связывание

Основным понятием ООП является объект, который в Delphi представляет собой переменную нового структурированного типа, описываемого с помощью ключевого слова Class. (Объекты всегда распределяются в куче, переменная типа класс это указатель)

Переменная типа Class под одним именем объединяет данные различных типов (поля) и процедуры их обработки (методы)

Объединение данных и процедур их обработки, называемое инкапсуляцией, с приданием ему свойств наследования и полиморфизма оказалось удивительно плодотворным

Любой класс может быть порожден от другого класса.

Класс Tobp порожден от класса Tobr.

Tobr – «класс родитель», а Tobp – «класс-потомок».

Порожденный класс (Tobp) автоматически наследует поля и методы своего родителя (в приведенном примере a, b, F1) и обогащает их новыми (c, d, e, P1).

Принцип наследования позволяет эффективно использовать уже наработанный задел программ и на его основе создавать классы для решения все более сложных задач. Это приводит к тому, что программный продукт представляет собой ветвящееся дерево классов.

Прародителем всех классов в Delphi является класс TObject. Все компоненты Delphi представляют собой созданные разработчиками классы.

Свойство полиморфизма позволяет использовать одинаковое название метода для решения сходных но несколько отличающихся у разных родственных классов задач.

Например, метод Add (добавить) имеется у многих компонент Delphi, хотя в каждой компоненте реализуется по разному.

Обеспечивается это тем, что в классе-потомке метод переписываются по новому алгоритму, т.е. перекрывается. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода с разными алгоритмами.

Obp1.Add; Obr1.Add; действуют по разному

Это и называется полиморфизмом объектов.

Полиморфизм можно организовать по-разному: используя раннее связывание метода с полями объекта, и позднее связывание.

Статические методы: характеризуются тем, что связывание метода с полями объекта осуществляется во время компиляции (раннее связывание). Предыдущие листинги иллюстрируют статические методы.

Виртуальные и динамические методы: связываются с объектом во время выполнения программы (позднее связывание).

Если метод объявлен виртуальным или динамическим, то нельзя менять типы и число параметров.

Позднее связывание методов позволяет реализовать казалось бы удивительную способность родительского метода использовать методы своих потомков.

Замещаемый одноименный метод родителя объявляется как динамический или виртуальный с помощью ключевых слов

Procedure Ris;dynamic; (или virtual)

В потомке замещающий метод объявляется директивой

Procedure Ris; override;

Вызов перекрытого метода родительского класса в одноименном методе потомка достигается с помощью зарезервированного слова

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

27. Что такое виртуальный и динамический способы реализации полиморфизма. В чем их различие. Понятие абстрактного метода. Как реализуется вызов родительского метода в методе потомке?

Виртуальные и динамические методы: связываются с объектом во время выполнения программы (позднее связывание).

Если метод объявлен виртуальным или динамическим, то нельзя менять типы и число параметров.

Позднее связывание методов позволяет реализовать казалось бы удивительную способность родительского метода использовать методы своих потомков.

Встретив объявления dynamic или virtual, компилятор создает таблицы соответствия DMT и VMT.

В этих таблицах помещаются адреса точек входа методов.

Таблица VMT “своего” класса хранится в каждом экземпляре объекта в особом, скрытом от программиста поле.

Таблица DMT хранится в VMT.

При каждом обращении к методу компилятор вставляет в соответствующую таблицу код, позволяющий извлечь затем из нее адрес точки входа в подпрограмму

Отличие таблиц DMT и VMT в том, что

DMT содержит адреса только тех методов, которые объявлены как dynamic в данном классе

VMT содержит адреса всех виртуальных методов данного класса: как нововведенных, так и унаследованных от родителей.

Любой класс имеет VMT, т.к. в Delphi все классы являются потомками класса Tobject, который в свою очередь содержит виртуальные методы.

Более того VMT родителя повторяется в VMT наследника с добавлением в нее сведений о виртуальных методах наследника, если таковые имеются

Использование таблиц VMT обеспечивает более быстрый поиск требуемого метода, но они и занимают много памяти при больших проектах. При использовании динамических методов таблицы DMT создаются только для тех классов, которые их имеют, т.е. они экономят память.

Var ColrBack:Tcolor;

Type

Tviz=class(Tobject)

Canvas:Tcanvas;

ColrLine:Tcolor;

x,y,r:word;

Procedure Ris;virtual;abstract;

Procedure Draw(bl:boolean);

procedure Show;

procedure Hide;

procedure MovTo(dx,dy,dr:integer);

end;

Метод Ris в родительском классе ничего не делает

Объекты абстрактного класса не создаются!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

28. Назначение конструктора и деструктора. Приведите примеры их описаний. Их вид в Delphi

Как всякая динамическая переменная, объект перед началом работы с ним должен быть создан конструктором класса TObject Create:

<Имя-переменной-типа-класс> := <тип-класса> . Create;

После окончания работы с объектом, выделенную под него память необходимо освободить деструктором класса TObject Destroy или Free:

<Имя-переменной-типа-класс> . Free;

в состав любого пользовательского класса могут быть введены свои методы Сonstructor и Destructor с дополнительными к Create и Free функциями.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

29. Понятие свойств класса, для чего они нужны? Как они описываются? Приведите примеры классов со свойствами

Правила хорошего тона при ООП требуют: обращение к полям должно осуществляться посредством методов.

Но каждый раз писать имя метода при обращении к полю не всегда удобно, поэтому придумали специальный механизм, регулирующий доступ к полям.

Свойство – это специальный механизм класса, регулирующий доступ к полям.

Свойства объявляются с помощью ключевых слов property… read…write…;

var p:Tip;

property сp:Tip read методr write методw.

Свойство связано с некоторым полем и указывает:

методr – метод чтения содержимого поля

(например, вывод на печать)

методw - метод записи в это поле.

Пример описания свойства

Type

MyСlass=class( )

p:Integer;

//получить значение p

Function Getpole:Tpole;

Begin

Getpole:=p; //отразить p на форме

label1.caption:=inttostr(p);

end;

// присвоить полю значение

Procedure Setpole(p0:Tpole);

Begin

p:=p0

end; //cp – имя свойства

Property cp:Integer read getpole write Setpole;

(Property cp:Integer read p;)

(Property cp:Integer write Setpole;)

...

end;// MyСlass=class

Работа со свойством

Var

Myobj:Myclass;xt:integer;

begin

....

Myobj:= Myclass.Create;

...

Myobj.cp:=10;

Myobj.Setpole(10);//это эквивалентно

Myobj.p:=10; // так можно но не рекомендуется

....

xt:=Myobj.cp;

xt:=Myobj.Getpole;//это эквивалентно

xt:=Myobj.p; //так можно но не рекомендуется