1

Алгоритм - это строгая, четкая последовательность операций, приводящая к решению задачи. точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату.

Алгоритмизация — это математический термин, определяющий процесс создания алгоритмов для решения каких-либо задач.

Пример алгоритма : Алгоритм "Как открыть дверь?"

**1.**Достать ключ из кармана.**2.**Вставить ключ в замок**3.**Повернуть ключ .**4.** Вынуть ключ.

**Свойства алгоритма 1. Массовость** – алгоритм должен описывать круг однотипных задач, исходные данные которых могут изменяться в определенных пределах. **2. Детерминированность** – это обусловленность всех шагов алгоритма потребностью решения данных задач. Свойство детерминированности выражается в том, что при заданных значениях параметров алгоритм выполняется формально, т.е. строго выполняется последовательность действий до появления результата.**3. Понятность** – предписания алгоритма должны быть сформулированы так, чтобы они понимались одинаково разработчиком и исполнителем, т.е. они должны быть однозначно понятны.**4. Дискретность** – четкое разделение всего пути решения задачи на отдельные этапы (шаги) так, чтобы ход выполнения алгоритма проходил поэтапно, вовремя корректируя действия исполнителя.**5. Результативность**– точное выполнение предписаний алгоритма должно привести к результату за n шагов, если правильно разработана исходная модель и сам алгоритм

2

Способы описания алгоритмов.Существуют следующие основные способов описания алгоритмов: словесное описание, псевдокод, блок-схема, программа.

*Словесное описание*– структура алгоритма на естественном языке. Например, инструкции по эксплуатации электроприборов. Никаких правил составления словесного описания не существует. Запись алгоритма осуществляется в произвольной форме на естественном языке. Данный способ описания алгоритмов не нашел широкого распространения.*Псевдокод*– описание структуры алгоритма на естественном, частично формализованном языке, позволяющее выявить основные этапы решения задачи, перед точной его записью на языке программирования. В псевдокоде используются некоторые формальные конструкции и общепринятая математическая символика. Строгих синтаксических правил для записи псевдокода не существует. Возможны различные псевдокоды, отличающиеся набором используемых конструкций и слов.*Блок-схема*– описание структуры алгоритма с помощью геометрических фигур с линиями-связями, показывающими порядок выполнения отдельных инструкций. Данный способ имеет ряд преимуществ. Благодаря наглядности, он обеспечивает «читаемость» алгоритма и явно отображает порядок выполнения отдельных команд. В блок-схеме каждой формальной конструкции соответствует определенная геометрическая фигура или связанная линиями совокупность фигур.*Программа*– описание структуры алгоритма на понятном языке алгоритмического программирования.

Пример алгоритма:

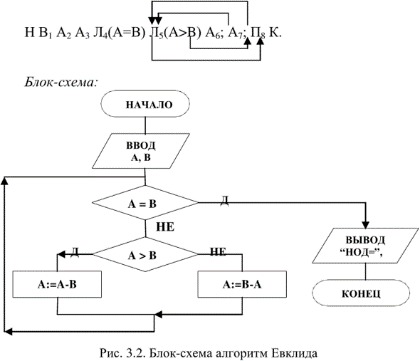
Чтоб создать документ, надо включить компьютер, открыть Word, создать новый документ, заполнить его текстом и прочим, сохранить

3

Операторная схема - аналитическая форма представления алгоритма с помощью операторов, отражающих содержание этапов.

Порядковый номер оператора, независимо от его типа, обозначается индексом. Операторы записываются в строку, для пояснения схемы переходов после логических операторов ставятся сверху или снизу горизонтальные стрелки, указывающие место перехода. Если операторы выполняются в естественном порядке, между ними не ставится разделитель, если после i-ro оператора (1+1)-й не выполняется, между ними ставится разделитель - точка с запятой. Для логических операторов возможно указание условия, по которому программа разветвляется на ветви.

Операторный метод дает наглядное логическое представление алгоритма, однако он был вытеснен графическим способом, который оказался более компактным и наглядным Арифметический оператор (вычисление) обозначается в русской транскрипции буквой А, логический - буквой Л, ввод данных -буквой В, печать - буквой П, начало - буквой Н, конец - буквой К. пример



4

*Схема алгоритма*– описание структуры алгоритма с помощью геометрических фигур с линиями-связями, показывающими порядок выполнения отдельных инструкций. Данный способ имеет ряд преимуществ. Благодаря наглядности, он обеспечивает «читаемость» алгоритма и явно отображает порядок выполнения отдельных команд. В блок-схеме каждой формальной конструкции соответствует определенная геометрическая фигура или связанная линиями совокупность фигур

Можно выделить такие особенности, которые должны быть у любой схемы алгоритма:

1)Обязательно должно присутствовать два блока – «Начало» и «Конец». Причем в единичном экземпляре.

2)От начального блока до конечного должны быть проведены линии связи.

3)Из всех блоков, кроме конечного, должны выходить линии потока.

4)Обязательно должна присутствовать нумерация всех блоков: сверху вниз, слева направо. Порядковый номер нужно проставлять в левом верхнем углу, делая разрыв начертания.

5)Все блоки должны быть связаны друг с другом линиями. Именно они должны определять последовательность, с которой выполняются действия. Если поток движется снизу вверх или справа налево (другими словами, в обратном порядке), то обязательно рисуются стрелки. 6)Линии делятся на выходящие и входящие. При этом нужно отметить, что одна линия является для одного блока выходящей, а для другого входящей.

Пример схемы алгоритма:

Составим блок схему алгоритма ,который высчитывает произведение S двух чисел A и B ,введенных с клавиатуры



5

Граф-схема алгоритма-это конечный связный ориентированный граф, вершины которого соответствуют операторам, а дуги задают порядок следования вершин(операторов) алгоритма.

Основные свойства граф-схемы:

1.Состоит из конечного числа точек, называемых вершинами, которые соединяются стрелками.

2.В граф-схеме есть 2 особых узла: 1-входной, в который не входит ни одна стрелка,2- выходной, из которого не выходит ни одна стрелка

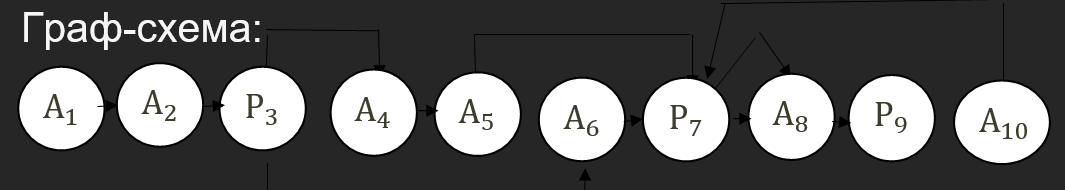
3. Все вершины граф-схемы отмечены своим оператором (А или Р)

4. Из каждого узла с символом А выходит только одна стрелка, из символа Р как минимум 2. Если в граф-схеме вместо операторов поставить их описания, то она превратиться в схему алгоритма

Пример:   


Где М-это количество указанных чисел меньших К  


1. М:=  0 ; Б:=  0
2. I:=1
3. Если  , тогда переходим к указателю 6
4. Б:= Б+1
5. Переход к указателю 7
6. М:=М+1
7. Если I⩾  n, тогда переход к указателю 9
8. I:=I+1
9. Переход к указателю 3
10. Конец



6

Численный метод – метод приближенного или точного решения математических задач ,основанный на построении конечной последовательности действия над конечным множесвтом чисел

Выбор численных методов:

1) исследовались вопросы получения погрешностей из-за непрерывных методов вычислений дискретными операциями

2)Начали изучать погрешности представленяи данных и накопления систематических ошибок при большом числе простейших операций над приближёнными числами

Выбирая численный метод необходимо проводить проверку правильности вычислений при решении задач на машине . Способ контроля при помощи повторных расчетов не всегда эффективен . Более экномным и эффективным может оказаться контроль путём проверок каких-лиюо заранее известных соотношений между высленными величинами

Сходящиеся итерационные методы-численные методы практически не требуют контроля правильности вычислений

Получение ошибочного результата при одной из итераций не приводит к ухудшению окончательного результата вычислений ,а лишь увеличивается количество итераций

Пример:

Необходимо вычислить sqrt(x) с точностью до 0.01 На ЭВМ корень считается процедурой ,которая выполняется много раз. Точность решения обычно задается значением максимально допустимой погрешности

7

|  |  |  |  |
| --- | --- | --- | --- |
|  | **P1** | **P2** | **P3** |
| **с** | **A1** | **A2** | **A3** |
| **C1** | **A1** | **A3** | **A1** |
| C1 | A3 | A2 | A2 |

**Маши́на По́ста** — абстрактная машина Поста состоит из бесконечной ленты, разделенной на разные секции, а также считывающую головку. В каждой секции м.б. ничего не записано либо записана метка, такая секция называется отличной. Информация о заполнении метками секции ленты характеризует состояние ленты.

Всего для машины Поста **существует шесть типов команд:**

1. xMy — поставить метку;
2. xCy — стереть метку;
3. x← y — сдвинуться влево;
4. x→ y — сдвинуться вправо;
5. х → y1|y2 — условная передача управления (если нет метки y1, если есть y2);
6. xCТОПx — конец программы.

*Пример*. Есть лента с записанными значениями ячейками, которые стоят группой, головка находится слева от этой группы на расстоянии нескольких секций. Необходимо очистить ленту и в конце, после группы, поставить одну метку.

1→2;

2 1,

3;

3С4;

4 5;

5 6,

3; 6М7; 7стоп7.

8

В контактных и логических схемах значения выходных переменных определяются только комбинацией значений переменных на входах в данный момент времени. Поэтому их называют *комбинационными схемами.* В более общем случае выходные переменные могут зависеть от значении входных переменных не только в данный момент, но и от их предыдущих значений. Иначе говоря, значения выходных переменных определяются последовательностью значений входных переменных, в связи, с чем схемы с такими свойствами называют *последовательностными.*Если входные и выходные переменные принимают значения из конечных алфавитов, то оба типа схем объединяются под названием *конечные автоматы.* Пусть *Xi -* алфавит входной переменной *хi,* а *Yi* – алфавит выходной переменной *yi*. Конечный автомат с *n* входами и *т* выходами характеризуется *входным алфавитом Х = Х1* ´ *Х2* ´ ... *Хn*и *выходным алфавитом Y = Y1* ´ *Y2* ´ ... *Ym*, причем символами входного алфавита служат слова *x* = (*x1*, *x2*, …, *xn*) длины *n*, а символами выходного алфавита - слова *y* = (*y1*, *y2*, …, *ym*) длины *т,* где *xi*Î *Xi*и *yi*Î *Yi*. Особого внимания заслуживают конечные автоматы с *двузначным структурным алфавитом,* зависимости между входными и выходными переменнымикоторых выражаются булевыми санкциями. Их значение обусловлено тем, что любая информация может быть представлена в двоичных кодах (двоично-десятичные коды чисел, телетайпный код в технике связи и т.п.). В то же время при технической реализации автоматов используются преимущественно двоичные элементы и двузначная логика.*Автомат называется автономным по входу, если его входной алфавит состоит из одной буквы: Х = {х}. Все входные слова у такого автомата имеют вид хх...х.*Из произвольного автомата с входным алфавитом *X = {х1,...хт}* может быть построено *т* различных автономных по входу автоматов исключением из графа переходов автомата всех ребер, кроме ребер с выбранной буквой х, (г' = 1,...,ш).*Аналогично, автомат называется автономным по выходу, если его выходной алфавит состоит из одной буквы Y - {у}.*Автономный по выходу автомат получается из произвольного автомата с выходным алфавитом *Y = {у{,у2,•??Ук]* исключением из графа переходов ребер со всеми выходными буквами кроме выбранной буквы *у..*

9

**Конечный автомат:** система, состоящая из конечного автомата, преобразующего символы алфавита **Р** в символы алфавита **С** в соответственной функции **Сi= F(Ci-1; Pi)** и преобразователя  λ, которая однозначно ставит символ в соответствии из алфавита **А. A=λ(Ci, Pi)** A1…A3 – случайные.

13

**Метасинтаксический язык Бэкуса.**

Метаязык, предложенный Бэкусом и Науром, впервые использовался для описания синтаксиса реального языка программирования Алгол 60. Наряду с новыми обозначениями метасимволов, в нем использовались содержательные обозначения нетерминалов. Это сделало описание языка нагляднее и позволило в дальнейшем широко использовать данную нотацию для описания реальных языков программирования. Были использованы следующие обозначения:

* символ " ::= " отделяет левую часть правила от правой;
* нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки "<" и ">";
* терминалы - это символы, используемые в описываемом языке;
* каждое правило определяет порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты "|".

Пример: <цифра> :: = 0|1|2|3|4|5|6|7|8|9; Формы Бекуса-Наура используются с двумя основными целями:

* они являются металингвистическим языком, стандартным для описания языков программирования;
* они описывают правила построения текстов или конструкций.

Таким образом, множество терминальных символов, множество нетерминалов с выделенными начальными символами и множество правил вывода входят в формальное определение порождающей грамматики. Язык, порождаемый такими грамматиками, это множество терминальных цепочек, которые можно вывести из начального

символа.

14

Начнём с нотации, введённой *Дж. Бэкусом* (J. W. Backus), которая называется *формой Бэкуса* — *Наура,* сокращённо *БНФ* (в английском оригинале — Backus Naur Form или BNF; как отметил *Д. Кнут*[110], эту аббревиатуру многие неправильно расшифровывают как «Backus Normal Form» — «нормальная форма Бонуса». Впрочем, изначально аббревиатура расшифровывалась именно так). БНФ была разработана Дж. Бэкусом для описания языка Алгол в сообщении о языке Алгол-60. Редактором этого сообщения был *П. Наур* [113].

БНФ представляет собой разновидность порождающей формальной КС-грамматики. В этой форме записи грамматики вместо символа «используется (читается «по *определению есть*»), а нетерминальные символы представляют собой произвольные последовательности букв (слова), заключенные в угловые скобки < >. Для БНФ также справедливо соглашение о том, что вертикальная черта «|» обозначает выбор одного из возможных правил.

Правила грамматики, записанные в БНФ, называются *металингвистическими формулами,* а символы «|», «<» и «>» — *метасимволами.*

Пример.

Пусть

def

*G* = ({?, *Т,* ?}, *{а,* +, \*, (,)},{?—>? + Т | *Т,* Г\*? | ?,?-^ (?) | а},?>. Запишем синтаксис языка с помощью БНФ:

* (?>:: = (?> + (Т) | *(Т)*
* (Т) *= (Г) \* (F)* I *(К)*

:: = «?» *а*

Рассмотрим БНФ, которая генерирует арифметические выражения, состоящие из однозначных чисел, скобок и знаков четырёх арифметических операций.

* (Выражение) :: = *(Число)* | *(Выражение) (Знак операции) (Выражение) ((Выражение))*
* (Знак *операции*):: = + | - | \* | /
* (Число) *:: =* 0 |1|2|3|4|5|6| 7| 8 | 9

У этого варианта имеется одно неоспоримое преимущество: его простота. Но эта простота может обернуться неприятными последствиями. Действительно, рассмотрим процесс порождения строки 5\*3-2 (стрелки здесь будут означать последовательные шаги):

* а) *(Выражение)*
* —>*(Выражение) (Знак операции) (Выражение)*
* —> *(Выражение)* \* *(Выражение)*
* —> *(Выражение)* \* *(Выражение) (Знак операции) (Выражение)*
* —> *(Выражение) \* (Выражение)* + *(Выражение)*
* —>...5\*3 + 2
* (многоточие означает, что нам безразлична последовательность порождения цифр).
* б) *(Выражение)*
* —> *(Выражение) (Знак операции) (Выражение)*
* —> *(Выражение) + (Выражение)*
* —> *(Выражение) (Знак операции) (Выражение)+ (Выражение)*
* —> *(Выражение) (Выражение)* + *(Выражение)*
* —>... —> 5\*3 + 2

Обе эти последовательности приводят к одному результату. Но первая из них сначала генерирует произведение, а вторая — сумму.

Допустим теперь, что эти деревья используются *программой-транслятором* не только для описания синтаксиса языка, но и для других целей, например, для перевода в постфиксную форму или для вычисления значения выражения.

Механизм вычисления выражения может быть таким: в процессе левого обхода дерева (операции выполняются слева направо) вычисляются промежуточные результаты для каждого поддерева, состоящего из трёх ветвей: двух операндов и знака операции. Тогда первое дерево будет соответствовать выражению: 5 \* (3 + 2) = 25, а второе — выражению (5 \* 3) + 2 = 17. Так как только второй путь приводит к верному результату, необходимо это учесть в БНФ: ввести в рассмотрение старшинство операций.

Исправленная БНФ выглядит следующим образом:

* (*Выражение***):: =***(Выражение)***+***(Терм) (Выражение) - (Терм) (Терм) (Терм***):: =***(Терм)***\****(Множитель) (Терм) / (Множитель) (Множитель) (Множитель)(Число) ((Выражение))*
* (Число) :: = 0 |1|2|3|4|5|6|7|8|9

Этот вариант БНФ уже позволяет порождать выражения, применяя правильный порядок применения правил, и тем самым однозначно вычислить значение любого порождённого ею выражения.

При всех своих достоинствах, БНФ имеет и недостатки.

Рассмотрим в качестве примера диалект TURBO Pascal, в котором запрещается использовать идентификаторы (имена переменных, типов и т. п.) длиной более 63 символов (вообще говоря, использовать их можно, но компилятор распознаёт идентификаторы по первым 63 символам, так что их использование бессмысленно). Мы хотим зафиксировать это в БНФ.

15

Синтаксические диаграммы Вирта(15 вопрос)

Синтаксические диаграммы были предложены Никлаусом Виртом для описания синтаксиса языка Pascal и являются удобной графической формой представления РБНФ.

Элементами синтаксических диаграмм Вирта являются: прямоугольники, кружки или овалы, стрелки. В прямоугольниках записываются имена металингвистических переменных, в кружках или овалах – основные символы языка, а стрелки определяют порядок сочетания металингвистических переменных и основных символов языка для образования определяемой синтаксической конструкции. Каждой синтаксической конструкции соответствует одна диаграмма Вирта. Имя определяемой синтаксической конструкции записывается над стрелкой, входящей в диаграмму (точка входа в диаграмму), которая, как правило, располагается в левом верхнем углу. Любой путь от точки входа в синтаксическую диаграмму к выходу (исходящая из диаграммы стрелка) представляет собой цепочку металингвистических переменных и основных символов языка, соответствующую одному из вариантов правой части РБНФ. На рис. 2.1 приведены синтаксические диаграммы, определяющие множество целых чисел.

Рис. 2.1. Синтаксические диаграммы для определения множества целых чисел

БНФ, РБНФ и синтаксические диаграммы Вирта дают возможность косвенно включать в формальное описание синтаксиса языков программирования элементы семантики, т. к. в них входят металингвистические переменные, являющиеся осмысленными названиями описываемых конструкций. При использовании автоматических методов анализа языков элементы семантики, заложенные в эти формальные модели, теряют смысл, поэтому в теории и практике проектирования языковых процессоров используются формальные грамматики.

Формальные грамматики и рассмотренные в разд. 2.1 формальные модели не позволяют описать синтаксис языка программирования полностью. Некоторые его аспекты, например контекстные условия, могут быть описаны только семантическими правилами.

В метаязыке диаграмм Вирта используются графические примитивы, представленные на рисунке 1.4.1.

При построении диаграмм учитывают следующие правила:

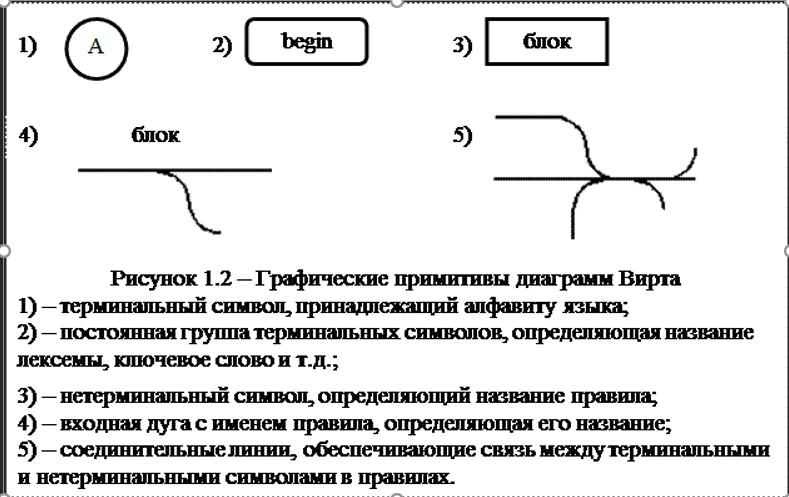
- каждый графический элемент, соответствующий терминалу или нетерминалу, имеет по одному входу и выходу, которые обычно изображаются на противоположных сторонах;

- каждому правилу соответствует своя графическая диаграмма, на которой терминалы и нетерминалы соединяются посредством дуг;

- альтернативы в правилах задаются ветвлением дуг, а итерации - их слиянием;

- должна быть одна входная дуга (располагается обычно слева или сверху), задающая начало правила и помеченная именем определяемого нетерминала, и одна выходная, задающая его конец (обычно располагается справа и снизу);

- стрелки на дугах диаграмм обычно не ставятся, а направления связей отслеживаются движением от начальной дуги в соответствии с плавными изгибами промежуточных дуг и ветвлений.



16

Алгоритмический язык – средство полного формирования вычислительных процессов для последующей реализации на ЭВМ. Основные задачи для которых созданы алг.яз: 1) ёАлгоритмы численного анализа (мат. расчеты, расчеты по формулам)

2)Процессы обработки данных 3)Обработка символьной информации.

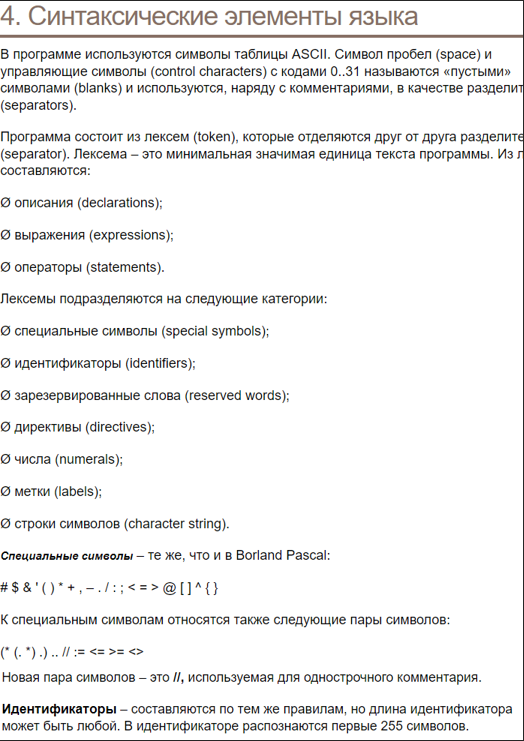
Этапы разработки:

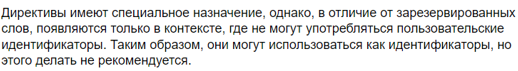
1. Отбор изобразительных средств языка – связан с анализом класса задач, которые будут решаться при помощи этого языка. Цель: выделить в задаче устойчивые структуры и действия над ними.
2. Написание языка – это значит задать правило, которое определяет синтаксис, помогая создать семантику.
3. Разработка транслятора – разрабатываются трансляторы, которые воспринимают текст на алгоритмическом языке и осуществляют его обработку и выполнение на машине.

17

Свойства хорошего языка программирования:1) Низкий порог вхождения. Языки с понятным Синтаксисом можно освоить гораздо быстрее и писать на них обычно гораздо приятнее.2) Логичный, лаконичный и понятный. легкочитаемый синтаксис и компактные программы помогут сэкономить время при разборе или написании кода \ поиске ошибок.3) Кроссплатформенность подходит для разных платформ: и Linux, и Windows.4) Широкое применение. Используется для разработки веб-приложений, игр, удобен для автоматизации, математических вычислений, машинного обучения, в области интернета вещей. Существует реализация под названием Micro Python, оптимизированная для запуска на микроконтроллерах (можно писать инструкции, логику взаимодействия устройств, организовывать связь, реализовывать умный дом).5) Высокая востребованность на рынкетруда.Любой хороший ЯП будет востребован на рынке труда и люди ,его освоившие могут не волноваться ,что в ближайшей преспективе у них будут финансовые трудности или же проблемы с трудоустройством.6) Высокая производительность.7) Бесплатность.Чем доступнее ЯП для кампаний и соло-разработчиков ,тем сильнне его комьюнити и больше обучающих видео,решений задач и проблем ,написанных на этом ЯП.

18





20

Проведём эксперимент. Введём в Google-переводчик фразу «Стыдно быть несчастливым» (это из Александра Володина). Переведём, например, на самоанский, Самоа — это в Океании. Программа выдала «Maasiasi e le fiafia». Звучит красиво. А теперь поменяем местами колонки и переведём фразу обратно на русский. Получилось «Стыд не весело». Справедливое утверждение, но ведь у нас было совсем другое!

Почему программа-переводчик не справилась с задачей?

**Ответ**

Потому что слова в языке многозначны, одно и то же можно выразить по-разному, с каждым переводом смысл чуть меняется.

Естественные языки, которые постепенно возникли в ходе истории человечества, а не создавались искусственно, не подчинены строгим правилам. В них одно слово может иметь множество значений в зависимости от окружающих его слов, ситуации и даже интонации. Это окружение слова, влияющее на его смысл, будем называть **контекстом**.

В естественном языке одну и ту же мысль можно выразить разными способами. Приглашение поехать в выходные на рыбалку может звучать и как «А не порыбачить ли нам в выходные?», и как «Давайте в субботу-воскресенье на рыбалку съездим!».

А вот в языках программирования такие вольности недопустимы. Если мы хотим на языке Python попросить компьютер получить данные с клавиатуры, преобразовать в целое число и записать в переменную n, мы должны написать n = int(input()) — только так и никак иначе, и при этом не потерять ни одну скобочку. В противном случае мы увидим сообщение о синтаксической ошибке. Язык программирования — **формальный язык**, он регламентирован строгими правилами.

**Формальный язык** — язык, в котором значение каждого слова или знака, правила построения предложений и понимания их смысла однозначны.

Вот несколько примеров формальных языков:

* запись арифметических выражений:  173 + (13 · 5) : 3
* язык записи шахматных партий: 1. e2-e4 e7-e5 2. Kg1-f3 Kb8-c6

Формальными языками являются и нотная запись, и дорожная разметка, и  навигационные знаки. Эти и другие формальные языки — удобные инструменты, которые упрощают жизнь людей.

21

На сегодняшний день имеется достаточно много различных определений понятия «база данных», от достаточно неформальных, описательных, до более-менее строгих. Так, К. Дж. Дейт в определяет базу данных следующим образом:**База данных** — это некоторый набор перманентных данных, используемых прикладными программными системами какого-либо предприятия.Под перманентными данными Дейт понимает данные, которые, будучи однажды помещены в базу данных, сохраняются там до тех пор, пока не буду удалены при помощи явно сформулированного запроса. Этим они отличаются от временных данных, которые возникают в процессе работы приложений и существуют только в течение сеанса работы, после которого исчезают.В качестве более формального определения можно привести использующееся в Гражданском кодексе Российской Федерации:**База данных** — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ).Данное определение уже в явном виде указывает на то, что базы данных – это совокупности данных, предназначенные для обработки с помощью вычислительной техники, что отличает их от других совокупностей данных, например, бумажных картотек, каталогов, справочников и так далее. Кроме того, необходимо отметить и указание на то, что материалы должны быть специальным образом систематизированы. Таким образом, очевидно, что для того, чтобы использовать базы данных, необходимо располагать соответствующим программным обеспечением, аппаратными средствами и так далее. Это приводит нас к необходимости ввести дополнительное понятие для описания совокупности данных и средств для их обработки – банк данных.**Банк данных** – это система специальным образом организованных данных (баз данных), программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

22

**Основные требования к базам данных *Отсутствие дублирования (избыточности) данных***, обеспечивающее однократный ввод данных и, соответственно, простоту корректировки. Под дублированием данных в базе понимают повторение одних и тех же описательных, то есть *неключевых* данных в разных массивах (таблицах).Например, включение адреса поставщика не только в справочник поставщиков, но и в массив поставок, в котором поставки определяется идентификаторами поставщика, поставляемого товара и датой поставки. Соответственно, в справочнике поставщиков данные о поставщике представлены однократно, а в массиве поставок они будут повторяться во всех поставках одного поставщика.Однократность ввода данных означает, что одинаковые описательные справочные данные вводятся и корректируются однократно. Кроме того, при вводе идентификационных данных их не приходится вводить в связанные логически массивы повторно.***Целостность и непротиворечивость данных*** в базе – означает такое наполнение базы данными, при котором все записи из разных массивов имеют корректные логические связи с записями других массивов, в случае если такие связи определены в логической структуре БД.Например, в БД, обладающей такими свойствами, не может сохраняться запись поставки, если в справочнике поставщиков отсутствует запись соответствующего поставщика, а в справочнике товаров отсутствует запись соответствующего товара.***Возможность многоаспектного доступа*** – означает, что обеспечиваются всевозможные выборки из массивов не дублированной информации и многоцелевое использование одних и тех же данных в различных задачах и приложениях пользователя.***Защита и восстановление данных*** при аварийных ситуациях, аппаратных и программных сбоях, ошибках пользователя,***Возможность модификации структуры*** базы данных без повторной загрузки данных. Т.о. база данных по мере углубления разработки может легко расширяться и модифицироваться

23

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности.Основные задачи:1. Обеспечение хранения в БД всей крайне важной информации.2. Обеспечение возможности получения данных по всем необходимым запросам.3. Сокращение избыточности и дублирования данных.4. Обеспечение целостности данных (правильности их содержания): исключение противоречий в содержании данных, исключение их потери и т.д. **Концептуальное (инфологическое) проектирование**— построение семантической (смысловой) модели предметной области. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных. Термины «семантическая модель», «модель базы данных», «модель предметной области», «концептуальная модель», «концептуальная модель базы данных», «концептуальная модель предметной области» и «инфологическая модель» являются синонимами.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER(entity-relation)-диаграммам.

Чаще всего концептуальная модель базы данных включает в себя:

· описание информационных объектов, или понятий предметной области и связей между ними;

· описание ограничений целостности, ᴛ.ᴇ. требований к допустимым значениям данных и к связям между ними.

24

**Модель "сущность–связь"**

Средством моделирования предметной области на этапе концептуального проектирования является модель "сущность–связь". Часто ее называют ER-моделью (Entity – сущность, Relation – связь). В ней моделирование структуры данных предметной области базируется на использовании графических средств – *ER-диаграмм* (диаграмм "сущность–связь"). В наглядном виде они представляют связи между сущностями.

Основные понятия ER-диаграммы – сущность, атрибут, связь.

*Сущность* - ϶ᴛᴏ некоторый объект реального мира, который может существовать независимо. Сущность имеет *экземпляры*, отличающиеся друг от друга значениями атрибутов и допускающие однозначную идентификацию. *Атрибут* - ϶ᴛᴏ свойство сущности. К примеру, сущность КНИГА характеризуется такими атрибутами, как автор, наименование, цена, издательство, тираж, количество страниц. Конкретные книги являются экземплярами сущности КНИГА. Οʜᴎ отличаются значениями указанных атрибутов и однозначно идентифицируются атрибутом "наименование". Атрибут, который уникальным образом идентифицирует экземпляры сущности, принято называть *ключом*. Может быть *составной* ключ, представляющий комбинацию нескольких атрибутов.

Предположим, что проектируется база данных, предназначенная для хранения информации о деятельности некоторого банка. Этот банк имеет филиалы. Филиалы управляются менеджерами. Клиенты имеют в филиалах счета разных типов – текущие, срочные, до востребования, депозитные, карточные. Филиалы обрабатывают эти счета. Описываемую предметную область назовем **БАНК**. В ней бывают выделены четыре сущности: филиал, менеджер, счет, клиент.

На ER-диаграмме сущность изображается прямоугольником, в котором указывается ее имя. К примеру,



В реальном мире существуют связи между сущностями. *Связь* представляет взаимодействие между сущностями. Она характеризуется *мощностью*, которая показывает, сколько сущностей принимает участие в связи. Связь между двумя сущностями принято называть *бинарной*, а связь между более чем с двумя сущностями – *тернарной*.

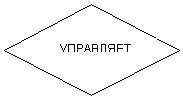
В рассматриваемой предметной области **БАНК** можно выделить три связи.

1. МЕНЕДЖЕР – УПРАВЛЯЕТ – ФИЛИАЛ

2. ФИЛИАЛ – ОБРАБАТЫВАЕТ – СЧЕТ

3. КЛИЕНТ – ИМЕЕТ – СЧЕТ

На ER-диаграмме связь изображается ромбом. К примеру,



Важной характеристикой связи является *тип* связи (*кардинальность*). Рассмотрим типы связей **1–3**.

Так как менеджер управляет только одним филиалом, то каждый экземпляр сущности МЕНЕДЖЕР может быть связан не более чем с одним экземпляром сущности ФИЛИАЛ. В этом случае связь **1**имеет тип "один-к-одному" (1:1). На рисунке 3.3 представлена ER-диаграмма для связи типа 1:1.



**Рисунок 3.3 – ER-диаграмма связи 1:1**

Так как филиал обрабатывает несколько счетов, а счет обрабатывается только одним филиалом, то каждый экземпляр сущности ФИЛИАЛ может быть связан более чем с одним экземпляром сущности СЧЕТ, а каждый экземпляр сущности СЧЕТ может быть связан не более чем с одним экземпляром сущности ФИЛИАЛ. В этом случае связь **2** имеет тип "один-ко-многим" (1:М). На рисунке 3.4 представлена ER-диаграмма для связи типа 1:М.



19

|  |
| --- |
|  |

Интерпретаторы и компиляторы. Для того чтобы процес­сор мог выполнить программу, эта программа и данные, с которыми она работает, должны быть загружены в опера­тивную память.

Итак, мы создали программу на языке программирова­ния (некоторый текст) и загрузили ее в оперативную па­мять. Теперь мы хотим, чтобы процессор ее выполнил, одна­ко процессор «понимает» команды только на машинном языке, а наша программа написана на языке программиро­вания. Как быть?

Необходимо, чтобы в оперативной памяти находилась программа-переводчик (транслятор), автоматически пере­водящая программу с языка программирования на машин­ный язык. Компьютер может выполнять программы, напи­санные только на том языке программирования, транслятор которого размещен в оперативной памяти компьютера.

Трансляторы языков программирования бывают двух ти­пов: интерпретаторы и компиляторы. Интерпретатор — это программа, которая обеспечивает последовательный пе­ревод инструкций программы на машинный язык и их вы­полнение. Поэтому при каждом запуске программы на вы­полнение эта процедура повторяется. Достоинством интерпретаторов является удобство отладки программы (по­иска в ней ошибок), так как возможно пошаговое ее выпол­нение, а недостатком — сравнительно малая скорость вы­полнения.

Компилятор действует иначе, он переводит весь текст программы на машинный язык и сохраняет его в исполни­мом файле (обычно с расширением ехе). Затем этот уже го­товый к выполнению файл, записанный на машинном язы­ке, можно запускать на исполнение многократно. Достоин­ством компиляторов является большая скорость выполне­ния программы, а недостатком — трудоемкость отладки, так как невозможно пошаговое выполнение программы.

Современные системы программирования, и в том числе Visual Basic, позволяют работать в режиме как интерпретато­ра, так и компилятора. На этапе разработки и отладки про­граммы используется режим интерпретатора, а для получе­ния готовой исполняемой программы — режим компилятора.

Процесс выполнения программы. Рассмотрим процесс выполнения программы на примере рассмотренной выше программы (проект «Переменные»), написанной на языке программирования Visual Basic.

Ввод текста программы в оперативную память. Текст программы вводится в оперативную память с помо­щью клавиатуры или считывается из внешней памяти. Текст программы займет в памяти определенное количество ячеек (например, с ячейки номер I по ячейку I+K).

Перевод программы на машинный язык. Наша про­грамма будет записана в памяти во внутреннем представле­нии языка программирования Visual Basic, который процессор «не понимает». Для перевода программы на машинный язык, понятный процессору, в памяти должна находиться программа-транслятор языка Visual Basic. Программа-транс - лятор после считывания в оперативную память из внешней памяти будет занимать в памяти определенное количество ячеек (например, с ячейки номер N по ячейку N+M).

Выполнение программы. После запуска программы на выполнение процессор последовательно будет считывать из памяти операторы и их выполнять.

В процессе выполнения оператора объявления переменных Dim в оперативной памяти для их хранения отводится необхо­димое количество ячееек: для целочисленных переменных intA, intB — по две ячейки, для неотрицательной целочис­ленной переменной bytC — одна ячейка, для переменной оди­нарной точности sngD — четыре ячейки, для переменной двойной точности dblE — восемь ячеек, для строковых пере­менных strA и strB количество ячеек, равное количеству символов, составляющих их значения, для логических пере­менных ЫпА, ЫпВ, ЫпС — по две ячейки. Таким образом, в памяти для хранения данных (значений переменных) будет отведено определенное количество ячеек, например ячейки с 1-й по 39-ю

Далее, в процессе выполнения операторов присваивания в отведенные переменным области оперативной памяти за­писываются их значения. Если в правой части оператора присваивания находятся арифметические выражения, то предварительно вычисляются их значения.

Затем с помощью метода Print производится вывод зна­чений переменных на форму, реализующую графический интерфейс программы. В этом процессе значения перемен­ных считываются из памяти и высвечиваются на экране мо­нитора.

10

В таблице по вертикали записываются все состояния системы, а по горизонтали все символы.

Чтобы определить текущее состояние находим предыдущее состояние по вертикали и по горизонтали.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **P1** | **Pn** | … | **Pe** | … | **Pr** |
| **С1** | **C3** | **C8** |  |  |  |  |
| **С2** | **C1** | **Cn** |  |  |  |  |
| **…** |  |  |  |  |  |  |
| **СJ** |  |  |  | **CJe** |  |  |
| **…** |  |  |  |  |  |  |
| **Сk** |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  | **P1** | **P2** |
| **С1** | **С2** | **С3** |
| **С2** | **С1** | **С2** |
| **С3** | **С4** | **С3** |
| **С4** | **С2** | **С1** |

Пример:

**Ci-1 = C1**

**P = P1**

**Ci = C3**

Автономный автомат представляет собой один столбец из основной таблицы

|  |  |
| --- | --- |
|  | **Pс** |
| **С1** | **С2** |
| **С2** | **С1** |
| **С3** | **С4** |
| **С4** | **С2** |

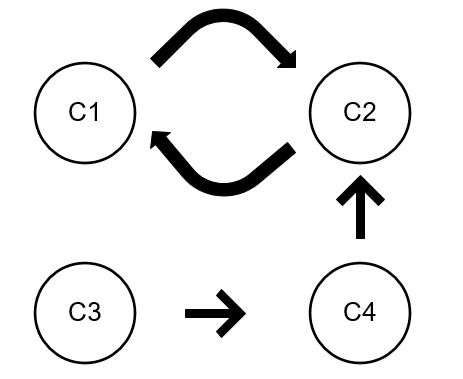
11

**Граф автомата** - это ориентированный связный граф, вершины которого соответствуют состояниям, а ребра графа – переходам между ними. Две вершины графа автомата соединяются ребром, направление указывается стрелкой. Ребру графа приписывают соответствующие значения входных и выходных сигналов, если они определены. Если переход из состояния aiв состояния aJ происходит под воздействием нескольких входных сигналов, то соответствующему ребру (ai, aJ) присваивают все значения входных и выходных сигналов.

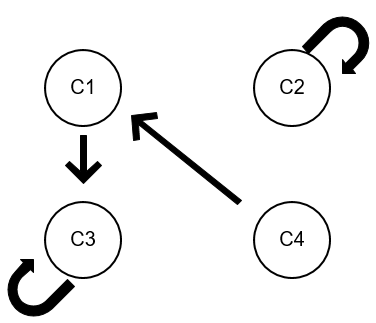
Вершина графа – состояние; Дуга - переход

A(P1)

|  |  |  |
| --- | --- | --- |
|  | **P1** | **P2** |
| **С1** | **С2** | **С3** |
| **С2** | **С1** | **С2** |
| **С3** | **С4** | **С3** |
| **С4** | **С2** | **С1** |



A(P2)



Объединенные графы автомата

12

Каждому графу автономного автомата можно однозначно сопоставить квдратную матрицу состоящую из **0** и **1**. **1**- есть связь между вершинами

**0**-нет связи между вершинами.

