Міністерство освіти та науки України
Криворізький державний педагогічний університет

Комп'ютерне моделювання та інформаційні технології в природничих науках

Збірник наукових праць

Кривий Ріг
Видавничий відділ КДПУ
2000

ББК 32.973.3
К 63
УДК 681.3.001.57+37.01:007

Комп'ютерне моделювання та інформаційні технології в природничих науках: Збірник наукових праць. – Кривий Ріг: Видавничий відділ КДПУ, 2000. – 464 с.

Збірник містить статті з різних аспектів застосування моделювання у природничих науках та освітній діяльності, нових технологій навчання фізики, математики та інформатики. Значну увагу приділено інформаційним технологіям в управлінській діяльності.

Для студентів вищих навчальних закладів, аспірантів, наукових та педагогічних працівників.

Затверджено Вченою Радою Криворізького державного педагогічного університету (протокол №8 від 18.04.2000 р.)

ISBN 966-7048-05-7

О ВЫБОРЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ ДЛЯ НАЧАЛЬНОГО ОБУЧЕНИЯ

А.П. Полищук, С.А. Семериков, Н.В. Грищенко
г. Кривой Рог, Криворожский государственный педагогический университет

Сравнение языков программирования – тема, всегда вызывавшая множество противоречивых суждений и споров. Если мы задаем вопрос, какой язык лучше, то прежде всего следует уточнить: лучше для чего, для какой сферы применения?

Существует множество публикаций, в которых предпринимались попытка оценить языки по определенному набору критериев. При этом за основу берутся, как правило, экспертные оценки. Однако, оценивая тот или иной язык, хорошо бы опираться на *объективные* критерии, а не мнение эксперта, пусть даже и являющегося признанным авторитетом в своей области.

Для начала определим сферу применения: обучение программированию. Отсюда естественным образом возникает и критерий – *простота* языка, выбираемого в качестве учебного, причем желательно, сравнивая различные языки, ответить на вопрос: на сколько рассматриваемый язык проще других, дав как качественную, так и количественную оценку его сложности.

Не существует общего критерия, вычисляемого в качестве оценки «сложности языка программирования». Тем не менее в любом языке, начиная с Алгола-60, есть строго формализованная часть – описание синтаксиса языка, что позволяет нам заняться оценкой объема этих описаний.

Синтаксические единицы любого языка одновременно являются и основными смысловыми понятиями: модуль, класс, блок, функция, процедура, метод, оператор, выражение. Ну а что, как не система понятий и их количество, может характеризовать сложность языка? Поэтому *в качестве меры сложности языка программирования будем рассматривать количественные характеристики формализованного описания его синтаксиса*.

Предметом нашего рассмотрения будут следующие языки: Алгол-60, Паскаль (в версии Н. Вирта), Си (K&R), Ада, Модула-2, ряд версий Турбо (Борланд)–Паскаля, Объектный Паскаль

(Дельфи), Си++, Оберон, Оберон-2, Java.

Алгол-60. Это первый язык, синтаксис которого был описан формально с помощью специально для этого созданной нотации – формул Бэкуса–Наура (ФБН). В 60-х – начале 70-х Алгол был самым популярным языком в СССР. Сейчас его не используют, но он рассматривается, так как, по сути, является родоначальником всех обсуждаемых языков.

Паскаль. Прямой потомок Алгола. При описании синтаксиса Паскаля его автор Н. Вирт использовал ФБН, добавив в нотацию скобки { и }, означающие повторение заключенной внутри них конструкции. Паскаль в оригинальной авторской версии не содержит средств раздельной компиляции – модулей, разнообразных числовых типов, строк переменной длины и многого из того, что добавлено в известные реализации.

Си. Этот язык достаточно традиционен. Приобрел популярность благодаря остроумным решениям, сделавшим запись программы на Си весьма компактной. Не накладывая на программиста особых ограничений, он дает ему максимальную гибкость. При описании языка использована нотация, в принципе эквивалентная ФБН. Каноническим считается описание, данное в книге создателя языка Д. Ритчи.

Как показали проведенные измерения, Си – не простой, а очень простой язык, недаром он завоевал такую популярность. Си весьма гармоничен, и хотя поощряемый им стиль принимают не все, но в изяществе языку не откажешь – что стоит только знаменитое ++. Интересно, кстати, что Си обладает одним из самых обширных наборов терминальных символов.

Ада. Официальный язык программирования американских военных. Происходит от Паскаля, но заметно сложнее его. Описан очень точно и строго. Для описания синтаксиса использован вариант ФБН.

Модула-2. Язык, который должен был заменить Паскаль, устранив основное его ограничение – отсутствие модульности. Но полноценной замены не получилось. В тот момент, когда нужно было обеспечить модульность в своих системах программирования, компания Borland решила, что выгоднее не переходить на Модулу-2, а добавить новые элементы в Паскаль. Тем не менее, известно, что Модула-2 использовалась и используется в

проектах, где важнейшую роль играет надежность. Средства межмодульного контроля Модулы-2 заметно совершеннее аналогичных возможностей Турбо-Паскаля и Си.

При описании синтаксиса языка Н. Вирт опирался на так называемые расширенные формулы Бэкуса–Наура (РФБН). В нотацию введены средства выражения итерации и задания необязательных частей, что позволяет в большинстве случаев обходиться без рекурсивных определений. Идентификатор в РФБН определяется так:

идентификатор = буква { буква | цифра }.

Фигурные скобки показывают, что заключенной в них конструкции может вовсе не быть либо она может повторяться сколько угодно раз. Получается гораздо проще и удобней.

При наших измерениях синтаксические правила для всех исследуемых языков были преобразованы именно в РФБН.

Турбо-Паскаль и Объектный Паскаль. Компилятор Турбо-Паскаль, разработанный Андерсом Хейльсбергом, был выпущен в продажу фирмой Borland в 1983 г. Эта версия уже содержала расширения языка, хотя и небольшие. В последующих выпусках расширений становилось все больше: встроенная «черепашья» графика (версия 3.0), от которой потом отказались, модули (4.0), средства ООП (объектно-ориентированное программирование; версия 5.5) и т.д. Начиная с версии 7.0, язык стал называться Borland-Паскаль, а с появлением системы Дельфи был переименован в Объектный Паскаль. Сейчас входной язык Дельфи по сравнению со стандартным Паскалем содержит очень много синтаксических расширений.

Линия Borland берет начало от Паскаля Н. Вирта, но дальше версии Турбо – Borland – Объектного Паскаля становятся сложнее и сложнее: избрав однажды путь расширения старого языка с сохранением обратной совместимости, его можно только усложнять, отказаться от чего-либо уже невозможно. Недаром по линии Borland идет монотонный рост значений всех без исключения критериев.

В результате из Паскаля получился язык, приближающийся по сложности к языку Ада. По большому счету Паскаль в версии Борланда (Inprise, а теперь уже и Corel) уже не может считаться общемировым языком программирования. Это фирменный язык

одной не очень большой американской компании. В таком смысле он ничем не отличается от Бейсика, языка другой, правда более крупной, фирмы. Отсутствие переносимости даже делает не вполне правомерным сравнение этого языка с Си, Си++, Java, Модулой, Обероном и Адой.

Си++. Первоначальное название «Си с классами». В язык Си Бъярн Страуструп ввел средства ООП и ряд других добавлений. До последнего времени это – самый популярный язык, хотя он и сложнее Си практически вдвое.

Оберон. Разработан Н. Виртом в 1987 г. Представляет собой существенно упрощенный вариант Модулы-2, в который добавлены расширяемые записи – основной механизм ООП.

Оберон-2. В 1992 г. были приняты расширения языка Оберон, предложенные коллегой Н. Вирта Ханспетером Мёссенбёком. В язык введены так называемые связанные процедуры – аналог виртуальных методов в других языках.

Java. Самый молодой и самый «модный» язык. Являясь непосредственным наследником Си++, отличается от него тем, что в нем устранены любые не относящиеся к ООП средства. Судя по вычисленным критериям, Java не только не может считаться простым, но является одним из самых сложных языков, более сложным, чем Си++, и вдвое более сложным, чем Оберон. Вопреки пропаганде, Java содержит мало чего-либо действительно нового.

Описание синтаксиса рассматриваемых языков в РФБН можно найти во множестве общедоступных источников. Там, где возможно, нами были использованы описания ISO/IEC (the International Organization for Standardization and the International Electrotechnical Commission) – организации, занимающейся выработкой международных стандартах в различных областях.

Правила, записанные в РФБН, как и текст на языке программирования, состоят из отдельных элементов – *лексем*. Лексемами являются определения понятий, называемые в теории формальных языков нетерминальными символами или просто *нетерминалами*. Например, в правиле

последовательность операторов = оператор {";" оператор}.

нетерминалами являются последовательность операторов и оператор. *Терминальные* символы – это те знаки, из которых и со-

стоит в конечном счете (terminal – конечный, заключительный) программа. При записи в РФБН терминальные символы ставятся в кавычки. В приведенном примере один терминал – «;». Терминальные символы – это тоже лексемы РФБН. Наконец, к числу лексем относятся специальные знаки, используемые в самих РФБН. В правиле о последовательности операторов это знак равенства, фигурные скобки и точка в конце.

Общее число лексем в описании синтаксиса языка может служить обобщенной характеристикой размера этого описания. Число лексем использовать в качестве меры объема гораздо лучше, чем, скажем, число знаков в описании. В этом случае значение нашего критерия не будет зависеть от того, на каком языке (русском, английском) или какими конкретно словами названы нетерминалы – понятия языка.

Число различных нетерминалов – следующая характеристика, которую мы будем определять путем вычислений. Количество используемых для описания языка понятий – несомненно важнейшее свойство, ведь именно от него зависит легкость освоения этого языка. Можно заметить, что число нетерминалов должно быть равно числу правил в описании синтаксиса, поскольку для каждого понятия должно быть ровно одно правило.

Набор и количество различных терминальных символов языка, упомянутых в синтаксических формулах, характеризуют *лексику* языка – набор знаков и специальных символов.

Во всех обсуждаемых нами языках существуют служебные слова, которые могут употребляться только в строго определенном смысле. Подсчет количества служебных слов позволяет оценить объем заучиваемых элементов языка.

Результаты анализа обобщены в таблице.

Несмотря на простоту, Оберон и Оберон-2 не получили широкого распространения, Java и Ада, в свою очередь, являются слишком сложными, а Алгол-60 и Модулу-2 можно считать вымершими языками. Поэтому реальными претендентами для дальнейшего рассмотрения являются языки Паскаль в редакции Никлауса Вирта (и его объектные расширения, предложенные Андерсом Хейльсбергом) и Си в редакции Денниса Ритчи (и его объектное расширение C++, предложенное Бьярном Страуструпом).

Таблица. Количество элементов в языке

Язык	*Лексемы*	*Нетерминалы*	*Терминалы*	*Служебные слова*
Алгол-60	1085	119	92	25
Паскаль	1012	110	84	35
ТП 2	1184	124	87	42
ТП 5	1331	127	87	48
ТП 5.5	1410	135	87	52
ТП 6	1488	143	89	55
ОП	1825	180	90	83
Модула-2	887	70	88	39
Оберон	765	62	90	32
Оберон-2	726	43	91	34
Си	917	49	123	27
Си++	1662	126	131	47
Java	1771	174	121	48
Ада	2206	226	102	63

Рассматривая синтаксис Паскаля и Си, можно сделать следующие выводы:

1. Даже в варианте Н. Вирта количество служебных слов в языке Паскаль больше, чем в языке Си (35 против 27). В последней необъектной его реализации (Турбо-Паскаль 5) ключевых слов уже 48, что превышает количество ключевых слов в объектном Си++!

2. Объектный Паскаль, используемый в среде разработки Дельфи, содержит 83 (!) ключевых слова, что делает его практически непригодным для целей обучения. В то же время язык Си++ в аналогичной среде визуального программирования Си++ Билдер содержит всего 47 ключевых слов.

3. Количество лексем в ТП5 на 45% больше, чем в языке Си, в Объектном Паскале – на 10% больше, чем в С++.

4. Для описания синтаксиса ТП5 используется 127 правил (терминалов), для описания Объектного Паскаля – 180 правил. В то же время весь синтаксис Си может быть задан всего 49 правилами, а синтаксис Си++ – 126. Таким образом, описать синтаксис объектного языка Си++ оказывается легче, чем описать синтаксис необъектного Паскаля.

5. Чем больше в языке нетерминалов, тем короче программы на этом языке, тем более компактна запись программ на этом языке. Из таблицы видно, что наиболее «многословный» язык – это Паскаль в нотации Н. Вирта, а самый краткий – даже не Си, а его наследник – Си++.

Сравнение синтаксиса языка дает нам объективный показатель его сложности – по сути, количества времени, затрачиваемого на изучение его синтаксиса, но ничего не говорит о его применимости к задачам обучения. По синтаксису Си оказывается более простым, чем Паскаль, но, возможно, Паскаль более применим для решения учебных задач, чем Си?

Для ответа на данный вопрос проведем содержательное сравнение этих двух языков, рассмотрев основные отличия и ограничения Паскаля по сравнению с Си. В качестве точки отсчета примем стандарт языка Си 1978 г. и стандарт языка Паскаль 1983 г., реализованные в компиляторах Турбо С 1.0 и Турбо Паскаль 1.0. Так как освоение ООП не входит в действующую школьную программу по информатике, мы не будем рассматривать объектные расширения этих языков.

1. Типы и области видимости

Паскаль – строго типизированный язык. Это означает, что каждый объект в программе имеет строго определенный тип, определяющий допустимые значения и операции над данным объектом. Язык гарантирует защиту от недопустимых значений и операций проверками на этапах компиляции и выполнения.

Строгий контроль типов не следует путать с анализом размерности. Если определяются типы `apple` и `orange` как

```
type
apple = integer;
orange = integer;
```

то вполне допустимо любое арифметическое выражение, включающее как `apples`, так и `oranges`.

Строгий контроль типов осуществляется различными способами. Например, аргументы функций и процедур проверяются на совпадение типов. В Паскале отсутствует свойственная Фортрану свобода в передаче параметров – так, попытка передать вещественное число в процедуру, ожидающую целое, вызовет ошибку на этапе компиляции.

Целые переменные могут быть объявлены так, что они могут

содержать только определенный набор допустимых значений; компилятор и проверка во времени выполнения гарантируют, что значения вне объявленного диапазона не будут занесены. Это похоже на услугу, хотя такая динамическая проверка вызывает большие накладные расходы.

Рассмотрим некоторые проблемы типа и области видимости.

1.1. Размер массива является частью типа

Если объявлены

```
var arr10 : array [1..10] of integer;
arr20 : array [1..20] of integer;
```

то `arr10` и `arr20` – массивы из 10 и 20 целых соответственно. Предположим, что мы хотим написать процедуру `sort` для сортировки целых массивов. Так как `arr10` и `arr20` имеют различные типы, становится невозможным написание процедуры, сортирующей и тот, и другой.

Во многих ситуациях просто неприемлемо использовать массив предопределенного размера – например, при написании программы сортировки строк переменного размера необходимо как можно плотнее заполнить память. Имея же фиксированный размер строки, мы сильно теряем в производительности.

Проблема привязки размера массива к типу до сих пор является самой большой проблемой Паскаля.

1.2. Связанные программные компоненты должны храниться отдельно

Так как Паскаль был задуман как однопроходный компилятор, язык требует, чтобы перед использованием процедуры и функции были объявлены. В результате типичная Паскаль-программа читается снизу вверх – все процедуры и функции на всех уровнях отображаются на экране до того, как они используются. Это противоположно естественному порядку разработки процедур и функций.

В известной мере это может быть смягчено механизмом, подобным `#include` в Си: исходные тексты могут быть включены в программу в тех местах, где это необходимо, не загромождая ее.

Существует также ключевое слово `forward`, позволяющее в Паскале отделить объявление заголовка функции или процедуры от ее тела; оно предназначено для определения взаимно рекурсивных процедур.

Когда позже объявляется тело, заголовок этого объявления может содержать только имя функции и не обязан повторять информацию из опережающего объявления.

Связанной с данной проблемой является та, что Паскаль имеет строгий порядок, в котором он ожидает появления объявлений. Каждая процедур или функция состоит из:

`label` объявления меток, если имеются

`const` объявления констант, если имеются

`type` объявления типов, если имеются

`var` объявления переменных, если имеются

`procedure` и `function` – объявления, если имеются

```
begin
```

тело функции или процедуры

```
end
```

Это означает, что все декларации одного типа (типов, например) должны быть сгруппированы вместе для удобства компилятора, даже когда программист хотел бы держать вместе некоторые логично связанные вещи для лучшего понимания программы. Редко возможно определить декларацию, инициализацию и использование типа и связанных переменных вместе.

1.3. Отсутствие раздельной компиляции

Официальный стандарт языка Паскаль не разрешает раздельной компиляции. Теоретически, нужды в ней нет – если компилятор достаточно быстр (и если исходные тексты всех подпрограмм всегда доступны), можно все просто перекомпилировать. На практике, конечно, компиляторы никогда не бывают достаточно быстры, а исходные тексты зачастую недоступны.

1.4. Различные проблемы типа и области видимости

При передаче массива в процедуру объявление переменной требуемого типа является обязательным – нельзя объявить процедуру с такими формальными параметрами:

```
procedure add10 (var a : array [1..10] of integer);
```

Приходится придумывать имя типа, объявлять этот тип, и объявлять формальный параметр как переменную этого типа:

```
type a10 = array [1..10] of integer;
...
procedure add10 (var a : a10);
```

Вполне естественно, что декларация типа физически отделяется от процедуры, которая его использует. Практика изобретения имен типов полезна для типов, которые используются часто,

но является порочной для использованных единожды.

Приятно иметь декларацию `var` для формальных параметров функций и процедур; процедура несомненно указывает, что она собирается модифицировать аргумент. Но вызов программы не имеет средств объявления, что переменная должна модифицироваться: информация об этом находится только в одном месте – объявлении подпрограммы.

Другим незначительным дефектом является то, что массивы по умолчанию передаются по значению: эффектом этого является то, что каждый параметр массива обычно автоматически объявляется программистом как `var`. Если декларация `var` неумышленно опускается, это приводит к трудно обнаруживаемым ошибкам.

Паскалевская конструкция `set` выглядит хорошей идеей, обеспечивая удобство соглашений и некоторую свободу проверки типов. Например, множество проверок, подобных

```
if (c = blank) or (c = tab) or (c = newline) then ...
```

может быть переписано достаточно ясно и возможно более эффективно как

```
if c in [blank, tab, newline] then ...
```

Но на практике, множества в большинстве случаев бесполезны, так как размер множества сильно зависит от реализации. К примеру, естественно попытаться написать функцию `isalphanum(c)` (‘‘является ли `c` символом или числом?’’) как

```
{ isalphanum(c) – истинна, если c символ или число }
function isalphanum (c : char) : boolean;
begin
isalphanum := c in ['a'..'z', 'A'..'Z', '0'..'9']
end;
```

Но во многих реализациях Паскаля (включая оригинальную) этот код терпит неудачу, поскольку множества просто слишком малы. Соответственно, множества обычно лучше не использовать, если размер и скорость выполнения программы критичны.

1.5. Непреодолимые ограничения

В Паскале нет способа обойти механизм типизации – отсутствуют механизмы, аналогичный кастингу (приведению типов) в Си. Это означает, что в Паскале невозможно написать систему ввода-вывода, отличную от встроенной в язык поскольку нет способа для сообщения о типе возвращаемого объекта, и нет способа преобразования таких объектов в произвольный тип для

последующего использования.

2. Управляющие структуры

Недостатки управляющих структур Паскаля мелки, но многочисленны – вместо того, чтобы добить программиста, они режут его по кусочкам.

Отсутствует гарантированный порядок вычисления логических операторов `and` и `or` – ничего подобного нет при использовании `&&` и `||` в Си. Этот недостаток чаще всего проявляется в циклах:

```
while (i <= XMAX) and (x[i] > 0) do ...
```

– чрезвычайно опасный код в связи с тем, что не существует способа узнать, проверяется ли `i` перед `x[i]` или наоборот.

Между прочим, круглые скобки в этом коде обязательны – Паскаль имеет только четыре уровня приоритета операций оператора, с реляционными внизу.

Отсутствует оператор `break` для выхода из циклов. Это соответствует «одновыходной» философии, предложенной Н. Виртом для большей строгости концепции структурного программирования, но приводит к обходным маневрам или дублирующемуся коду, особенно в связи с невозможностью управления порядком, в котором вычисляются логические выражения. Рассмотрим эту общую ситуацию, выраженную средствами Си:

```
while (getnext(...)) {
if (something)
break
rest of loop
}
```

В отсутствии оператора `break`, первая попытка реализовать на Паскале будет следующей:

```
done := false;
while (not done) and (getnext(...)) do
if something then
done := true
else begin
rest of loop
end
```

Но это не работает, так как нет способа заставить ‘‘`not done`’’ вычисляться перед следующим вызовом `getnext`. Это приводит примерно к такому коду:

```
done := false;
while not done do begin
```

```
done := getnext(...);
if something then
done := true
else if not done then begin
rest of loop
end
end
```

Конечно, можно использовать `goto` и метку (которая, к тому же, должна быть предварительно объявлена) для выхода из цикла. В противном случае досрочный выход – головная боль, почти всегда требующая введения булевой переменной и изрядной доли хитрости. Сравните обнаружение последнего непробельного символа в массиве на Си:

```
for (i = max; i > 0; i = i - 1)
if (arr(i) != ' ')
break
```

и Паскале:

```
done := false;
i := max;
while (i > 0) and (not done) do
if arr[i] = ' ' then
i := i - 1
else
done := true;
```

Лишь в седьмой версии среды Турбо Паскаль (1992 г.) появилось средство для обхода этой ситуации – в модуль `System` были добавлена процедуры `break` и `continue`.

Инкремент в цикле `for` может быть только `+1` or `-1`, в Си – произвольный.

Отсутствует оператор `return` – функция возвращает значение присвоением его псевдопеременной, совпадающей с именем функции, но функция и после такого присвоения продолжает выполняться. Это иногда приходит к излишнему коду только для того, чтобы убедиться, что по всем ветвлениям в конце функции мы получаем корректное возвращаемое значение.

Оператор `case` в Паскале разработан лучше, чем в Си, исключая то, что в отсутствии секции, аналогичной `default`, поведение его становится непредсказуемым. В то же время практика программирования показывает, что конструкция `case` используется крайне редко.

3. Библиотеки

Run-time окружение Паскаля сравнительно бедно, и нет механизма его расширения, кроме, возможно, библиотек в официальном стандарте языка. Встроенный ввод-вывод Паскаля имеет заслуженно плохую репутацию – он «верит» только в записеориентированный ввод-вывод. У него также есть соглашение предварительного просмотра, которое слишком жестко, чтобы быть правильно реализованным в диалоговом режиме. В основном проблема состоит в том, что система I/O верит, что она должна прочитать одну запись впереди той записи, которая обрабатывается. В диалоговой системе это означает, что, когда программа начинается, первое первым ее действием должна быть попытка прочитать с клавиатуры первую строку ввода, перед любым действием, которое выполняется в начале программы. Но в программе

```
write('Please enter your name: ');
read(name);
...
```

опережающее чтение должно повесить программу, ожидая ввода до того, как будет напечатано приглашение.

Большинства таких эффектов можно избежать очень тщательной реализацией ввода-вывода, но это чревато дополнительными накладными расходами.

Паскалевская система ввода-вывода – тяжелое наследие исходной операционной системы, для которой Паскаль был разработан; даже Вирт признает, что это недостаток, хотя и не ошибка. Допускается, что текстовые файлы состоят из записей, представляющих собой строки текста. Когда последний символ строки прочитан, встроенная функция `eoln` становится истинной; в этой точке необходимо вызвать `readln`, чтобы начать чтение новой строки и восстановить состояние `eoln`.

Аналогично, когда последний символ файла прочитан, встроенная функция `eof` становится истинной. В обоих случаях, `eoln` и `eof` должны вызываться перед каждым чтением `read`, а не после. Это тоже является источником головной боли, когда необходимо вначале прочитать, а затем проверить, не закончился ли во время этого чтения файл.

Ввод-вывод языка Паскаль отражает концепции диалоговой системы CDC, для которой Паскаль был изначально разработан. Файловая переменная

`var fv : file of` тип

очень специфичный объект – он не может быть нигде ни присвоен, ни использован, за исключением вызова встроенных процедур `eof`, `eoln`, `read`, `write`, `reset` и `rewrite`.

`reset` и `rewrite` – процедуры, а не функции; они не возвращают состояние удачи-неудачи открытия файла, а просто завершаются аварийно. И переменная `fv` не может появиться в выражении вида

```
reset(fv, filename);
if fv = failure then ...
```

для указания дальнейших действий в случае неудачи открытия файла.

Поскольку в Паскале не возможно написать распределитель памяти общего назначении (нет способа сообщить тип объекта, который такая функция должна возвращать), язык имеет встроенную процедуру `new`, распределяющую пространство из кучи. Только определенные тип могут быть распределены, поэтому невозможно распределять, например, массивы произвольного размера для хранения символьных строк. Указатели, возвращенные `new`, могут передаваться, но не изменяться: над указателями не определены арифметические операции.

4. Косметические замечания

Большинство этих вопросов раздражают опытных программистов, а некоторые неприятны даже начинающим, но с ними можно жить. Паскаль, наряду с большинством других Алголподобных языков, использует точку с запятой как разделитель утверждения, а не терминатор (как это сделано в PL/I и Си). В результате необходимо усвоить сложные правила, когда ставить точку с запятой, а когда это неправильно. Рассмотрим фрагмент программы:

```
if a then
b;
c;
```

Но если что-то вставить перед `b`, точку с запятой ставить уже не нужно, потому что она теперь предшествует `end`:

```
if a then begin
b0;
b
end;
c;
```

Если мы сейчас добавим `else`, мы должны убрать точку с запятой перед `end`:

```
if a then begin
b0;
b
end
else
d;
c;
```

И так далее, и так далее – программа пишется, а точки с запятой бегают вверх-вниз по программе.

Существует общепринятый экспериментальный результат из психологии программиста: точка с запятой как разделитель в десять раз увеличивает вероятность ошибки, чем точка с запятой как терминатор. (В Аде, наиболее значимом языке, базировавшемся на Паскале, точка с запятой является терминатором.) К счастью, в Паскале можно почти всегда закрывать глаза и использовать точку с запятой как терминатор. Исключения – в таких местах, как объявления, где проблема разделителя и терминатора не является столь уж серьезной, да еще перед `else`, что легко запомнить.

Программисты на Си считают `begin` и `end` более громоздкими по сравнению с `{` и `}`.

Имя функции является и вызовом этой функции; нет способа отличить такой вызов функции от простой переменной кроме как знанием имен функций. Паскаль использует Фортрановский трюк, по которому имя функции подобно переменной в пределах функции, за исключением того, что в Фортране имя функции действительно является переменной, и может появиться в выражениях, а в Паскале появление его в выражении – рекурсивный вызов: если `f` – функция без аргументов, то `f:=f+1` – рекурсивное обращение к `f`.

Скудность операторов в Паскале, вероятно, связана со скудостью уровней приоритетов операций.

В языке отсутствует макропроцессор. `const` позволяет в 95% определить простые константы подобно `#define` в Си, но в более сложных он бесполезен. Встроить макропроцессор в компилятор Паскаля вполне возможно – это позволило бы сымитировать процедуру `error` как

```
#define error(s) begin writeln(s); halt(0) end
```

Вызов, подобный

```
error('little string');
error('much bigger string');
```

работает, так как `writeln` может обрабатывать строки любого размера.

5. Выводы

Завершая, сформулируем еще раз основные доводы, по которым использование Паскаля для начального обучения является менее предпочтительным, чем использование Си:

1. В связи с тем, что размер массива является частью типа, невозможно написание процедур общего назначении для обработки массивов произвольных размеров. В частности, обработка строк очень затруднена по сравнению с тем, как они обрабатываются в Си.
2. Однопроходная природа языка заставляет представлять процедуры и функции в неестественном порядке; осуществление разделения различных объявлений разбрасывает программные компоненты, которые логично должны быть вместе.
3. Отсутствует приведение типов и адресная арифметика, в связи с чем язык не отражает числовой природы данных в компьютере.
4. Двойственность использования имени функции в ее теле и использование точки с запятой не как терминатора увеличивает вероятность появления трудно обнаруживаемых ошибок.
5. Отсутствие раздельной трансляции препятствует разработке больших программ и делает невозможных использование библиотек, скомпилированных в объектные модули.
6. Порядок вычисления логических выражений не соответствует правилам логики.
7. Стандартный ввод-вывод дефективен и не расширяем, а библиотеки – бедны.

В своей исходной форме Паскаль – язык игрушечный, разработанный, вслед за Бейсиком, для начального обучения программированию. Однако многие исследования показали, что учебные языки со своей задачей справляются хуже, чем языки общего назначения, как в силу своей синтаксической сложности, так и в связи с присущими им ограничениями, препятствующими их применению для решения сколь-нибудь сложных задач.

Нам представляется бессмысленным обучать программированию на учебном языке – дальнейшее обучение вынуждает переходить на языки общего назначения.

Язык программирования, выбираемый в качестве языка для начального обучения, должен обладать рядом необходимых качеств, обеспечивающих легкость восприятия и приемлемую скорость освоения методов составления текстов программ:

- как можно более короткий словарь – чтобы основные усилия обучаемого сосредоточить на методической стороне программирования;
- словарь языка должен быть англоязычным, т.е. базироваться на языке, ставшим де-факто языком межнационального общения; обучение на национальных «суррогатах» – напрасная трата времени;
- учебный язык не должен содержать абстрактных типов данных, не совместимых в операциях с числовыми типами, например «символьный тип», «строковый тип», «булевский тип» и пр.; поддерживаемые в языке типы данных должны прямо отражать способность компьютера обрабатывать только числа и не вносить ненужной путаницы в сознание обучаемого между числовыми кодами и способами их трактовки и отображения;
- язык должен обладать функциональной полнотой профессиональных языков программирования.

Сегодня есть только один такой язык общего назначения – это язык Си (и его надмножество C++), сделанный в свое время инженером Деннисом Ритчи не для коммерческого распространения, а «для себя» и, наверное, поэтому получившийся столь удачным и для профессионального применения, и для начального обучения. Проведенный нами анализ показывает, что это язык отвечает объективным критериям синтаксической простоты, являясь самым простым из всех рассмотренных, и в то же время имеет полный набор конструкций, наиболее адекватных для процедурного программирования. В то же время традиционно используемый язык Паскаль в реализации фирмы Borland оказался по результатам тестирования не только одним из самых сложных, но и куда менее подходящим для задач обучения по сравнению с Си.

Зміст

Наукове видання

Комп'ютерне моделювання та інформаційні технології в природничих науках

Збірник наукових праць

Підп. до друку 03.04.2000 Формат 80х84 1/16.
Бумага офсетна №1 Зам. №4-0302
Ум. друк. арк. 26,10 Наклад 500 прим.

Видавничий відділ Криворізького державного педагогічного університету
КДПУ, 50086, Кривий Ріг-86, пр. Гагаріна, 54

E-mail: vyd@kpi.dp.ua