72-183

Стэнфордский университет
Отделение вычислительных наук

Никлаус Вирт

О НЕКОТОРЫХ ОСНОВНЫХ ПОНЯТИЯХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Стэнфорд, Калифорния
I мая 1967 г.

72-184

Развитие языков программирования в последнее время привело к появлению языков, чей рост обладает симптомами раковой опухоли: добавление новых элементов выходит из под какого бы то ни было контроля авторов языка, а природа этих новых образований часто оказывается несовместимой с существующим составом языка. Для того, чтобы освободить язык от таких симптомов, необходимо, чтобы он строился из основных концепций, обоснованно выбранных и независимо применяемых. Правила, управляющие языком, должны быть простыми, общеприменимыми и непротиворечивыми.

Для того, чтобы достичь такой простоты и непротиворечивости, фундаментальные концепции языка должны быть тщательно отобраны и определены с максимальной ясностью.

На практике это означает, что существует некоторый оптимум числа основных концепций, ниже которого не только реализуемость этих концепций на реальных машинах, но также и их значение для человеческой интуиции становится сомнительным из-за их слишком большой степени общности. Нижеследующие заметки не предлагают каких-нибудь готовых к употреблению решений, но я надеюсь, что они смогут пролить свет на некоторые примыкающие проблемы и внутренние трудности. Эти заметки предназначены суммировать и обобщить различные идеи, которые частично имеются в существующих языках, частично обсуждаются в Рабочей группе ИФИП 2.I и частично новые. Делая основной упор на выяснении концептуальных явлений в языках, нельзя в то же время игнорировать проблемы обозначений. Однако, в данном случае не предлагается какого-либо формального или законченного определения обозначений синтаксиса; концепции будут вместо этого в основном

иллюстрироваться на примерах, используя обозначения, столь близкие к АЛГОЛу, насколько это будет возможно.

I. О СТРУКТУРАХ ДАННЫХ

Элементарные концепции вычислительных процессов суть следующие:

- Существуют некоторые величины, называемые "значениями" и элементарные классы или типы (возможно только один) значений, между которыми существуют некоторые элементарные отношения. Эти отношения или отображения представляются в вычислительной машине ее операциями, которые порождают новую величину, новое значение (называемое результатом), находящееся в указанном отношении к данным значениям (называемыми операндами).

- Существуют ячейки (обычно называемые "переменными"), которые способны принимать значения и которые имеют имя. Это имя используется для упоминания хранимого значения.

- Существует оператор для присваивания нового значения некоторой ячейке.

Словарь, используемый для описания процессов, должен содержать по меньшей мере одно обозначение для каждого элемента множества значений, а также по меньшей мере одно обозначение для каждого отношения между значениями в каждом классе. В то время как множество элементарных значений обычно фиксируется в языке программирования, множество ячеек, вовлекаемых в некоторый процесс, является специфическим для данного процесса и должно быть определено в его описании. Следовательно, и имена, обозначающие эти ячейки, должны выбираться индивидуально (описываться). Необходимым правилом является, что либо имена ячеек должны быть отличимы от обозначений значений (и отношений), либо выбранные ячейки, оказавшиеся идентичными с некоторым значением

не могут больше явно использоваться для обозначения этого значения.

Существенно, что группы элементарных значений могут объединяться и затем рассматриваться как составные или структурные значения. Обычно такие значения обозначаются перечислением их компонент, разделенных некоторым разделителем (например, запятой) и выделяемых скобками. Имя ячейки, содержащей структурное значение, затем используется для того, чтобы обозначать всю совокупность величин компонент.

Эта концепция была некоторым простым способом реализована в языке ЭЙЛЕР [В и В]. Практичность этого решения однако в значительной степени сомнительна по следующим причинам:

I. Поскольку ячейка может хранить любое значение, в том числе и составное, физические размеры ячейки в терминах машинной памяти не фиксируются. Реализация такой схемы требует применения косвенной адресации и динамического распределения памяти в столь большом объеме, что это сокращает возможности применения языка во многих приложениях.

2. Слишком большой динамизм и отсутствие избыточности в языке затрудняют программисту проверять правильность написанной программы.

3. Допуская, что индивидуальные элементы составного значения могут упоминаться с помощью имени хранящей ячейки, вслед за которой стоят индексы, сразу можно понять, что то же самое обозначение может быть использовано и в операторах присваивания для изменения элемента в этой структуре. Поскольку присваивания могут быть сделаны только ячейкам (но не величинам) ячейка, хранящая составное значение, должна рассматриваться

72-187

как структурная ячейка. Отсюда вытекает, что создание ячеек становится в большой степени неявным действием, поскольку присваивание некоторой n-арной величины влечет за собой создание n ячеек. Из этого можно сделать вывод, что язык программирования может не содержать обозначения для структурных значений, но должен содержать обозначения для структурных ячеек. Позиционные отношения между значениями, тем самым, могут существовать и выражаться только в терминах структуры ячеек, содержащих эти значения.

1. Определения типов

Эти трудности могут быть преодолены приписыванием каждой ячейке фиксированной структуры в тот момент, когда эта ячейка вводится. Для практических целей это по существу не является ограничением, поскольку в большинстве приложений программа использует только сравнительно небольшое количество различных типов структур, в то время как многие использованные ячейки имеют одну и ту же структуру. Каждый может рассматривать данные элементарные классы или типы значений как имеющие некоторую элементарную или вырожденную структуру. Ячейка, тем самым, может быть описана как имеющая данный элементарный тип и, следовательно, может принимать значения только данного типа. В АЛГОЛе 60 это достигается с помощью описаний типа. Далее, более сложные структуры могут рассматриваться как композиции элементарных структур; и для того, чтобы приписать имя такой структуре, так же называемой некоторым типом, необходимо ввести новую конструкцию, называемую "определением типа". Это может

72-188

иметь форму, иллюстрируемую следующим примером: тип Лицо (Целый возраст; Логический пол; Вещественный вес).

"Лицо" - это имя новой структуры, которая состоит из трех элементов, называемых "полями", которые являются элементарными структурами, а именно - Целый, Логический и Вещественный, соответственно. Как видно, определение типа используется также для того, чтобы приписать имена индивидуальным полям, тем самым оно соответствует описаниям класса записи [В и Х]. Допускается, что элементарные типы вводятся с помощью фиксированных определений типа в данном окружении программы. Фактически элементарные типы обычно сами строятся из битов, и их подструктура зависит от конкретной реализации и конкретной машины.

Если конституенты сами могут иметь любой тип, то тогда достаточно считать, что определение типа состоит только лишь из линейного списка его конституент. Примеры: тип История болезни (Целое кровяное давление; состояние сердца; Логический диабет).

тип Пациент (Целый возраст; Логический пол; История болезни здоровье).

Иногда желательно давать числовые имена полям структур, с тем чтобы эти имена могли быть вычислены. Например, может существовать

тип А (Вещественный I, 2, 3, 4) ,

для которого мы немедленно можем ввести сокращение

тип А (Вещественный [I : 4])

без дальнейших пояснений. Такая структура называется одномерный массив, а имена полей называются индексами; все элементы этой структуры имеют один и тот же тип. Многомерные массивы,

72-189

чьи элементы обозначаются более чем одним индексом, могут быть определены следующим образом:

тип В (Вещественный I, 2, 3 | I, 2, 3, 4).

Вышеупомянутое сокращение приводит к следующей краткой форме

тип В (вещественный [I : 3] | [I : 4]).

Отличие этой структуры от структуры, определяемой как

тип С (А [I : 3])

состоит в том факте, что I2 элементов В имеют тип Вещественный, в то время как С состоит из трех элементов типа А, которые в свою очередь состоят из четырех элементов типа Вещественный. Если В рассматривается как матрица, то его строки и столбцы не имеют явного обозначения и рассматриваются на одном и том же уровне, в то время как С рассматривается как линейная структура, состоящая из строк.

Показанные обозначения делают явным сходство концепций массивов и записей [В и Х]. Оно позволяет автоматически вводить поля, имеющие структуру массива:

тип Счет (Целый номер; Вещественный баланс; А депозит).

2. Описание ячеек

Введение ячеек (переменных, записей) требует, чтобы они содержали указания на тип ячейки вместе с именем, которое приписывается новой ячейке.

Примеры:[ж)] новый (Целый) i

новый (А) а, ав

новый (В) в, в1, в2

ж) для того, чтобы облегчить чтение последующих примеров, имена ячеек начинаются со строчной, а имена типов с заглавной буквы.

72-190

новый (С) с

новый (Лицо) джек, джил

новый (Пациент) смит

новый (Счет) ас

Символ новый выбирается для того, чтобы указать, что вводится новая ячейка данного типа. Вместо новый можно также использовать слова ячейка или переменная, для того чтобы подчеркнуть создание ячейки или переменной. В терминах реализации эти описания всегда приводят к некоторому распределению памяти.

В АЛГОЛе 60

новый (Целый) i сокращается как

целый i

и это соглашение служит для всех элементарных типов. Если правила языка таковы, что на месте идентификатора типа может встретиться определение типа, то тогда пример

новый (A) a

может также иметь форму

новый (Вещественный [1:4]) a

или сокращенно

вещественный [1:4] a

из которого аналогия с АЛГОЛовским описанием массива

вещественный массив a [1:4]

становится очевидной.

3. Обозначения ячеек

В настоящее время существуют различные обозначения для ячеек и компонент структурированных ячеек:

72-191

α	β	γ	δ
а[2]	а.2	2 of а	2(а)
в[2,3]	в.2,3	2,3 of в	2,3(в)
джек[возраст]	джек.возраст	возраст of джек	возраст(джек)
смит[здоровье]	смит.здоровье	здоровье of смит	здоровье(смит)
с[2]	с.2	2 of с	2(с)
с[2[3]]	с.2.3	3 of 2 of с	3(2(с))
смит[здоровье[диабет]]	смит.здоровье.диабет	диабет of здоровье of смит	диабет(здоровье(смит))
ас[главный]	ас.главный	главный of ас	главный(ас)
ас[депозит[3]]	ас.депозит.3	3 of депозит of ас	3(депозит(ас))

В этом месте кажется подходящим проверить результаты предыдущей унификации концепций, а также сравнить результирующие обозначения с конструкциями, используемыми в существующих языках. Обозначения α совпадают с АЛГОЛом 60 в форме индексных переменных. β имеется в ПЛ/1 и в КОБОЛе (применяясь только к фиксированным, т.е. к невычисляемым именам). γ совпадает с обозначением указателей полей в [ВВ], и δ с этим же в [В и Х], применяясь в обоих случаях только в связи с невычисляемыми именами полей. Там, где разрешается применять вычисляемые имена, непременно должны быть разрешены выражения, что быстро в силу синтаксических требований исключает все случаи, кроме α. Для использования ячеек с буквенными, т.е. невычисляемыми именами полей, обозначения δ представляются более естественными в силу их аналогии с обычным обозначением для функций и предикатов, с которыми могут ассоциироваться имена полей. Можно заключить из предыдущего, что унификация однородных структур с вычисляемыми именами полей (индексы QMU) и неоднородных структур с невычисляемыми полями (идентификаторами) нежелательна, главным образом, в силу традиции обозначений.

72-192

Это тем менее желательно с точки зрения реализаций, поскольку вычисление индексов над массивом, состоящим из полей разной длины, представляет собой сложный и неэффективный процесс.

Относительно привлекательным решением этой дилеммы является а) допущение в качестве вычисляемых только числовых имен полей (индексов), б) заключение их в различимые скобки, в) использование привычного постфиксного обозначения (α) для индексов и префиксного обозначения (β) - для указателей полей с буквенными именами).

Примеры:

а[2]

в[2,3]

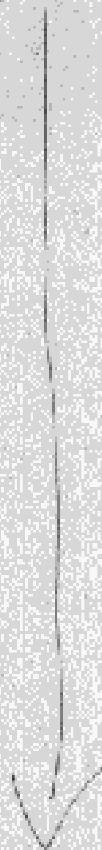
возраст(джек)

здоровье(смит)

с[2]

с[2[3]]

депозит(ас)[3]



72-193

4. Ячейки без явных имен.

До сих пор имело место предположение, что каждая ячейка, используемая в программе, явно обозначается именем, приписанным ячейке ее описанием. Однако в некоторых задачах обработки данных либо число требуемых ячеек априори не известно, либо нет необходимости, чтобы все ячейки были доступны от самого начала до конца процесса. Тем самым ~~[illegible]~~ желательно иметь средство создания ячеек
B+ в любое время (т.е. динамически).

Для любой созданной ячейки должно существовать средство ее упоминания. Поскольку ее имя не вводится в программу явно (т.е. в виде идентификатора), имена приходится рассматривать как объекты (ссылки), которые могут использоваться для упоминания ячеек. Тем самым создание ячейки не только влечет резервирование памяти, но также образует имя выделенной ~~[illegible]~~ ячейки. Это имя будет называться <u>ссылкой</u> и будет считаться элементарным значением типа Ссылка. Динамическое создание ячейки можно обозначить

r := Лицо

результатом выполнения чего является присваивание r ссылки на новую ячейку. Форма

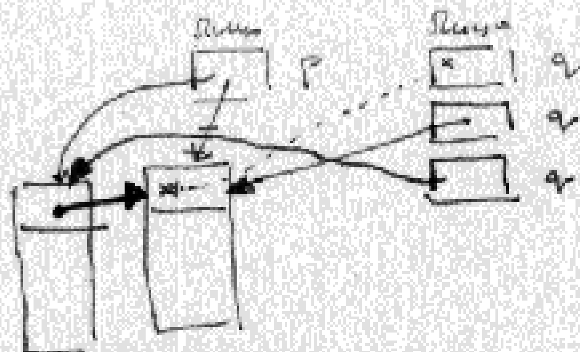
r := Лицо [21, <u>ложь</u>, 101.5]

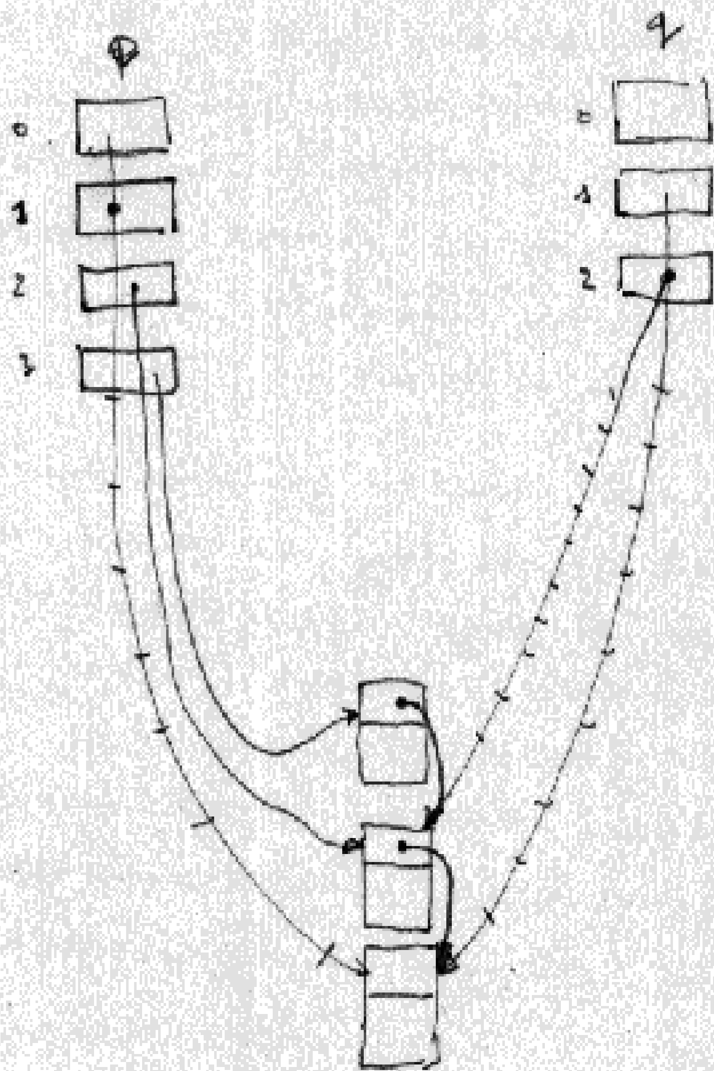
может быть применена, если вновь созданным полям ячейки должны быть одновременно присвоены начальные значения. Описание r обозначается через "<u>ссыл</u> r", что является сокращением формы

<u>новые</u> (Ссылка) r.

72-194

p := Sluso





72-195

Правило, которое требует, чтобы значение ссылки, присвоенное ссылочной ячейке, упоминало бы объекты только одного типа, создает значительные преимущества при реализации. Этот тип может быть указан в следующем виде в описании ссылочной ячейки.

<u>ссыл</u> [Лицо] r
<u>ссыл</u> [Целые] k

Причины привязывания ссылочных ячеек к фиксированным классам обсуждены в [В и Х].

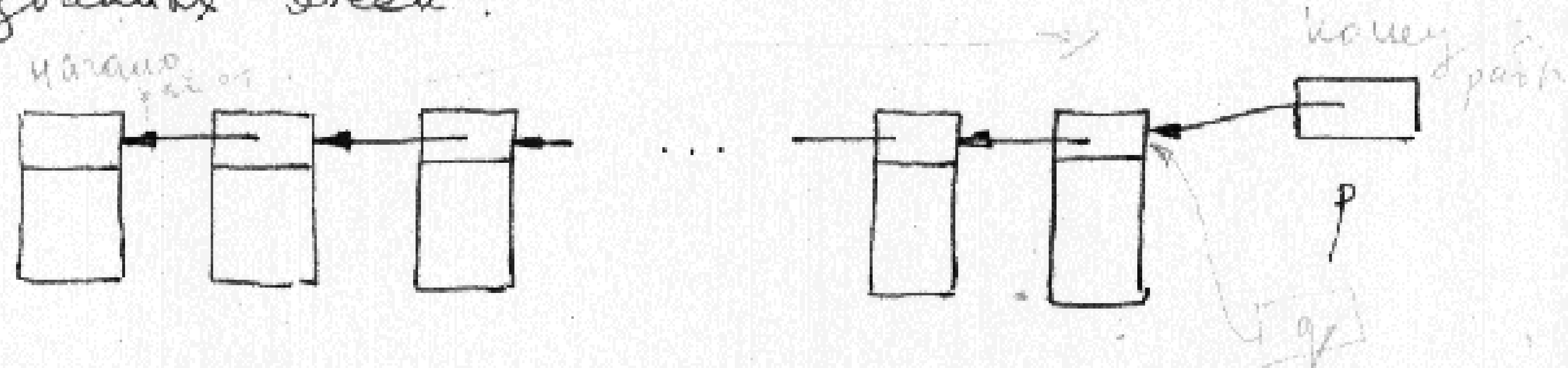
Надо подчеркнуть, что тип, фигурирующий в описании ссылочной ячейки, не обозначает подструктуры самой ссылочной ячейки, которая является элементарной, т.е. бесструктурной. Этот тип обозначает структуру упоминаемой величины.

Теперь можно использовать ссылки, чтобы выражать различные функциональные отношения между объектами, представляемыми ячейками. Если значениями поля f ячейки x являются ссылки и это поле упоминает ячейку y, то можно сказать, что y находится в отношении f к x. Ниже приводится пример куска программы, использующего механизм динамического создания ячеек:

<u>тип</u> Лицо (Ссылка [Лицо] сын; Целый возраст; ...);
<u>ссыл</u> [Лицо] p, q;
L: p := Лицо; сын(p) := q;
q := p; <u>на</u> L

72-196

Этот кусок программы описывает создание бесконечного числа ячеек типа Лицо. В точке L значение q упоминает самого "младшего" члена цепочки предков. Иллюстрация демонстрирует последовательность созданных ячеек:



Каждая стрелка представляет значение класса "Ссылка", которое хранится в поле этого типа, называемом "сын".

Вышеприведенный пример подсказывает также и способ для обозначения значения данных полей динамически созданных записей, находящихся в согласии с ранее введенными обозначениями. Если вместо имени ячейки, поле которой упоминается, появляется имя ссылочной ячейки, то тогда неявно подразумевается, что это обозначает поле косвенно упоминаемой ячейки.

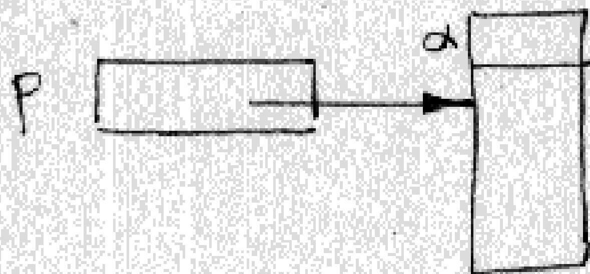
Пример:

возраст (джек) — джек = имя лица

возраст (p)
сын (p) — p = имя ссылки на лицо

Эти соглашения представляются совершенно естествен-

ними не вызывают никаких проблем, поскольку ссылочные ячейки сами не имеют структуры и поэтому обозначения полей единственным образом должны упоминать поле упомянутой ячейки. Однако возникает дилемма когда обозначается вся ячейка, а не одно из ее полей:



что обозначает p: ссылочное значение, упоминающее α, или само α? Ниже предлагаются два решения:

a. p обозначает ссылку на α, а Лицо(p) используется для обозначения α.

Тогда

p := q

обозначает копирование ссылки, в то время как

Лицо(p) := Лицо(q)

обозначает копирование содержимого ячейки типа Лицо.

в. Фактическое значение p определяется контекстом (например, соответствием типов), так что в

q := p

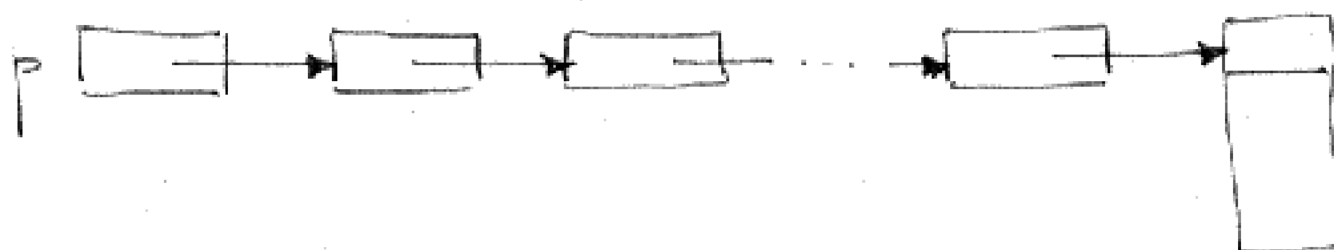
p обозначает ссылку на α, в то время как в

джек := p

p обозначает саму ячейку α типа Лицо.

72-198

Последнее решение, принятое в [BB], очевидно ведет к новым проблемам, когда p принимает значение, указывающее ячейки, которые сами имеют класс Ссылка.



Единственно, что можно в этой ситуации, это либо обозначать ссылочное значение, хранящееся в p, либо обозначать ячейку типа Лицо, находящуюся в конце цепочки ссылок. Никакого способа указать значения промежуточных ссылочных ячеек здесь нет. Кроме концептуальных сложностей, затрудняющих понимание таких конструкций в программе, возникают также обоснованные сомнения в практической полезности этих механизмов, что в совокупности подсказывает нам нецелесообразность динамического порождения ячеек типа Ссылка (и элементарных типов, вообще), в языках программирования. Другой аспект этой ситуации рассматривается в следующем разделе.

16 +

5. Блочная структура

Блочная структура была введена в Алгол 60 для ограничения областей действия имен (идентификаторов). Поскольку имена привязываются к величинам своими описаниями, а сами они не являются объектами, которыми можно манипулировать, ячейка становится

недоступной за пределами области действия ее имени. Это означает, что участок памяти, отведенный этой ячейке, отныне может использоваться для других целей.

Динамически создаваемые объекты не имеют имени, которое могло появиться в программе, и, тем самым, могут быть достигнуты только через внутренне создаваемые ссылки, так "срок жизни" не подчиняется ограничениям, вызываемым областями действия (в том же смысле, что и константы не имеют ограниченной области действия). Тем самым освобождение участка памяти, отводимого для динамически создаваемых ячеек, не может инициироваться выходом управления из данного блока, но может происходить лишь в некоторые непредсказуемые моменты времени, когда становится известным, что никакие ссылки на эту ячейку, ни явные, ни неявные, уже не могут быть значениями ссылочных ячеек, имеющих явно указанные имена (называемые "appellations" в [BB]). Однако, в силу

+

того, что описания ссылочных ячеек должны содержать спецификацию типа упоминаемых ячеек, эти описания не могут появиться вне области действия соответствующих определений типа. Следовательно существование ссылочных значений ограничивается областью действия типа упоминаемых величин, и все ячейки этого типа становятся недоступными с момента выхода из его области действия, что и может быть сигналом для освобождения памяти.

17 +

До сих пор ссылки (т.е. ссылочные значения) могли вводиться в дело только путем динамического

создания ячеек. В отношении рассматривались ссылки, упоминающие явно именуемые величины. Однако эта ситуация является неотъемлемой чертой как языка ЭЙЛЕР, так и преемника Алгола [ВВ] и поэтому требует особого рассмотрения.

В ЭЙЛЕРЕ ссылочное значение, упоминающее величину с именем x обозначается @x. В [ВВ] ссылка на величину x обозначается просто x; означает ли x значение величины x или ссылку на эту величину, определяется контекстом, примерно, по тем же правилам, по которым различаются случаи 'возраст(джек)' и "возраст(р)", рассмотренные выше. Это в принципе возможно, поскольку в отличие от ЭЙЛЕРА со всеми именованными величинами связываются фиксированные типы. Однако, если сохранить все допущения, сделанные в этом параграфе, (а именно, что ссылочные значения не подчиняются ограничениям области действия в отличие от явно указанных имен) то появляется противоречие. Это противоречие выступает в виде невероятных ситуаций, как показывает следующий пример, если только не прибегать к ревизии Алголовской блочной структуры:

```
начало ссыл [целый] k; целый j;
    начало целый i; i := 1;
α:          k := i
β:      конец;
γ:      j := k
конец
```

В α, в соответствии с тем фактом, что k имеет тип ссыл, k получает значение ссылки на ячейку i. В β происходит выход из области значения переменной i и, в соответствии с

72-201

[зачёркнуто] Алголовской традицией, у тебя нет памяти, зачитать i, освобождается. В [зачёркнуто] содержимое ячейки, упоминаемой значением k (пореченному i?), присваивается j. Кажется необходимым ревизовать определение Алгол 60 так, чтобы подчинить правилу копирующих только имена (   ), но не сами обозначаемые ими величины. Правда это, по-видимому, убивает самую идею блочной структуры. Фактически этот постулат эквивалентен тому, что вся память будет распределяться [зачёркнуто] так же, как и для динамически создаваемых ячеек.

Кажется, что единственно разумным решением было бы запрещение ссылочным ячейкам принимать в качестве значения ссылки на явно именованные величины.

Прежде чем обсуждать последствия этого ограничения, имеет смысл немного поговорить о реализации. Как уже говорилось, существуют ячейки как элементарных типов, так и составной структуры. Большинство машин в состоянии копировать и присваивать любое элементарное значение так же легко, как и ссылочное значение (адрес). Таким образом всегда разумно действовать с самой рассматриваемой величиной, а не со ссылкой на нее. Мало того, окончательный доступ к значению всегда будет более сложен, если он делается путем косвенной адресации. Естественно подумать, что язык должен максимально противодействовать использованию косвенной адресации в таких случаях.

72-202

С составными структурами, напротив, не так легко манипулировать, как со ссылками. Более того, поскольку размеры составных структур обычно могут быть вычисляемыми (массивы в Алголе 60), их размещение должно делаться динамически, что неизбежно приводит к их косвенной адресации. Из этого вытекает, что составные структуры неявно используют ссылочные ячейки, независимо от того, заботится об этом программист или нет. Описание массивов в Алголе 60 гораздо более точно описывается явными шагами

тип α (Вещественный 1:n);
ссыл [α] a;
a := α

нежели просто в виде

вещественный массив a[1:n],

и вхождение a в программу естественно понимать как обозначение ссылочной величины, указывающей на динамически созданные ячейки массива. Отсюда следует, что составные ячейки явно именовать нельзя, или, если это допускается в языке, это должно пониматься как сокращение в смысле приведенного примера. В том случае, когда определение типа помещается в том же блоке, что и описание ссылки (или если они в сокращенной форме задаются одновременно), то действие механизма распределения памяти одно и то же, что и в описаниях массивов Алгола 60: память освобождается при выходе из блока.

Эти соображения употребимости и в реализации элементарных и составных ячеек также полностью применимы к их рассмотрению как параметров процедур, несмотря на тот факт, что пропоненты ссылок на явно именованные

величин используют механизм параметров в своих мотивировках.

72-203

Квинтэссенция предыдущих трех разделов содержится в следующем:

- Ячейки элементарных типов всегда описываются и тем самым получают явные имена (   );
- Ячейки составных структур всегда создаются динамически, причем их структура задается явным определением типа.
- Ссылочные ячейки имеют элементарный тип и их значения всегда упоминают величины некоторого данного типа. Как следствие ссылки могут упоминать только составные ячейки.

18 +

Это исформулированное ограничение никак не ограничивает возможности языка, т.к. всегда можно определить структуру (или тип), состоящую только из одного поля.

6. "Исключение" ссылок

Если в языке подразумевается, что составные записи упоминаются всегда косвенно, то роль символа ссыл сводится лишь к тому, чтобы напоминать нам об этом соглашении. Фактически его можно опустить. Например, обозначение

19 +

ссыл [T] a, в

может быть заменено сокращением 72-204

T a, b

где T — идентификатор, введенный некоторым определением типа. Так сделано в языке АЭД [P]. Надо отчетливо понимать, что

a := b

означает копирование ссылки, а не самих упоминаемых структур. Некоторая вытекающая из этого путаница иллюстрируется нижеследующим куском программы, в котором последний оператор меняет не только возраст от p, но также и возраст от q:

```
новый (Лицо) p, q;
p := Лицо; возраст(p) := 10;
q := p;
возраст(p) := возраст(p) + 1
```

Использовать это сокращение или нет — это дело вкуса, то нам представляется оправданной такая точка зрения, что сосуществование обоих конструкций,

новый (T) r

и

ссыл [T] r ,

больше добавляет к концептуальной сложности языка, нежели к его полезности.

72 – 205

7. Ввод – вывод

Операции ввода-вывода — это присваивания значений (обычно составных), хранимых в памяти одного вида, ячейкам, находящимся в памяти другого вида. Если операции ввода-вывода составляют неотъемлемую часть языка, то тогда правила, определяющие эти операции, должны быть совместимы с правилами выполнения других действий. Простейший способ выбрать совместимые правила — это сделать их одинаковыми. Это значит, что данные, подлежащие вводу и выводу, должны описываться во "внешних" памятях так же, как они описываются во "внутренней памяти". Фактически спецификации вида памяти вместе с описаниями ячеек могут рассматриваться как примечания, зависимые от реализации.

Пример:

ссыл [T] a, b [диск]

ссыл [T] c, d [лента]

ссыл [T] x, y [память]

Присваивания вида

$x := a \qquad c := y$

20+

могут пониматься как операции ввода и вывода, соответственно. Заметим, что в этом случае не произойдет копирования только ссылок, т.к. предполагается, что ссылки указывают на ячейки только в заданном виде памяти.

Сведение операций ввода-вывода к простым присваиваниям может привести к тяжелым последствиям

72-206

и издержкам в использовании внешних видов внешней памяти, если только не сформулировать естественные ограничения на виды структур, учитывающие некоторые внутренние свойства внешних устройств.

Для того чтобы сформулировать такие ограничения, предлагается некоторая классификация структур:

- Структура, в которой число полей фиксируется определением типа, называется <u>статической структурой</u>.

- Структура, в которой все поля имеют один и тот же тип называется <u>векторной структурой</u>. (Именами ее полей обычно являются вычисляемые индексы.)

- Структура, которая не содержит полей, упоминающих другие структуры, называется <u>базисной структурой</u>.

Если присваивания требуют переноса информации между памятями разного вида и этот перенос нужно осуществить с минимумом административных затрат, то такие присваивания полезно ограничить массивами статической базисной структуры. Такое ограничение, естественно, оставляет за бортом ряд применений, тому, что обычно называют массивами записей (таблицами), т.е. линейных последовательностей записей данных, не содержащих перекрестных ссылок. Ограничение можно несколько ослабить, если просто потребовать, чтобы возможные ссылочные поля все были бы пустыми при присваивании.

72-207

Только что введенное понятие деки используется в смысле того, что более специально называется линейной декой или лентой, представляющими собой линейный массив старших структур (как выше). Дека, однако, отличается от более общих массивов ограничением, налагаемым на порядок доступа к ее элементам: с линейной декой связывается неявный индекс, указывающий на единственный доступный в данный момент элемент. Каждый обмен (частью) с декой автоматически увеличивает этот индекс на единицу. Существуют также некоторые стандартные операции над деками, которые позволяют изменять этот индекс.

Кажется, что это понятие деки является необходимым и достаточным для того, чтобы удовлетворительным способом включить в язык работу с устройствами памяти с органически последовательным обращением, такими, как ленты, построчная печать, перфораторы и читающие устройства.

8. Операции над заданными типами.

Алгол 60 содержит только такие операции, которые в этой работе называются операциями над элементарными типами. Эти операции встроены в вычислительную машину. Операции над значениями структурных типов обычно описываются в терминах последовательностей операций над компонентами. Способом для удобного сокращения таких последовательностей являются процедуры в Алголе 60. Можно получить модификацию этой концепции, распространив ее на обычные инфиксные знаки операций,

72-208

~~[crossed out]~~
шукси допущения, что ввиду эта операция применяется ~~[crossed out]~~ последовательно к конструентам ~~[crossed out]~~ структуры. Такая модификация называется "переопределением" (см. также [X]) и применяется к векторным структурам. В предыдущих разделах этот принцип уже применялся в операторах присваивания.

9 Резюме

- Существует ~~[crossed out]~~ фиксированный набор элементарных типов данных, не имеющих подструктур. Этот набор включает тип ссылка.

- Определение типа вводит типы структурных данных и связывает с ними (типами) ~~некоторое~~ обозначение (идентификатор). Структуры этого типа ~~[crossed out]~~ задаются либо в виде последовательности полей, причем каждому полю приписывается свой идентификатор и свой тип, либо в виде одно- или многомерного массива однородных элементов, которые ~~[crossed out]~~ обозначаются вычисляемыми индексами.

- Переменные, называемые здесь ячейками, имеют некоторый фиксированный тип, т.е. могут хранить значения только этого данного типа.

- Ячейки элементарных типов вводятся описаниями. Область действия их имени и их собственный срок жизни определяются блочной структурой.

72-209

Идея: разрезервирование имен, заменяемых структурным значением указателя присваиванием всей ячейке значения nil, при этом, если среди полей этой ячейки есть ссылочные поля, то они тоже аннулируют (так ли?) уничтожение упоминаемых ими ячеек.

- Ячейки структурных типов вводятся "динамически". Они не имеют ; вместо этого к ним обращаются косвенно посредством ссылки, которая является значением ячейки типа <u>ссылка</u>.

- Описания ячеек типа <u>ссылка</u> всегда указывают тип ячеек, которые ссылка может упоминать.

- Если ссылка r упоминает ячейку C типа T, то обозначения r и T(r) применяются для обозначения собственно ссылки и упоминаемой ячейки C, соответственно. Тем самым обозначения ячеек не зависят от контекста.

- В описаниях ссылочных ячеек можно указывать вид памяти, содержащей упоминаемые ячейки.

72-211

II. О СТРУКТУРАХ ПРОГРАММ

1. Операторы и выражения

Фундаментальным понятием структур программ является понятие оператора (присваивания). Оно обозначает некоторое "законченное действие", позволяющее считать, что результат выполненных действий полностью определяется значениями ячеек, участвовавших в процессе. Выполняя программу с помощью бумаги и карандаша можно выписать в явном виде все промежуточные результаты каждого оператора. Концептуально это наиболее важная ситуация, которая имеет особое значение для тех, кто сталкивается с проблемой проверки правильности программ. В Алголе выполнение оператора (присваивания) состоит в вычислении некоторого выражения, и затем в присваивании вычисленного "результата одной или нескольким ячейкам. (Заметим, что все операторы Алгола, исключая "неоператор" на, по существу могут быть сведены к операторам присваивания или к их последовательностям.) Выражение является такой составной частью, которая вычисляется с помощью обрывка бумаги (для промежуточных результатов) и тот факт, что этот обрывок бумаги может быть выброшен после выполнения каждого оператора, находит свое отражение в синтаксисе, гласящем, что <выражение> может быть конституентом <оператора>, но не наоборот. Этот порядок нарушен в Алголе тем фактом,

что в качестве конструкта в выражении можно использовать процедуру-функцию, где тело состоит из операторов. Отсутствие же этих явных указаний в синтаксисе компенсируется «правилом копирования», согласно которому <оператор> становится конструктом <выражения>. Последствия этой ситуации в свое время многократно и горячо обсуждались, сейчас объединены названием «побочных эффектов». В той мере, в какой влияют на ясность программ, побочные эффекты вредны; однако в некоторой ослабленной и контролируемой форме они могут быть весьма полезны. А если некоторое средство полезно в некоторых случаях, его очень трудно игнорировать ради сохранения принципа, сколь разумным бы он ни был.

Тем не менее, возникает вопрос: либо побочный эффект должен входить в язык как его неотъемлемая часть, либо его нужно полностью исключить.

Первое решение реализуется ликвидацией различия между операторами и выражениями и с рассмотрением выражения присваивания

$$v := e$$

как тождественной операции над e и с побочным эффектом, состоящим в присваивании v значения e. Эта философия была принята в ЭЙЛЕРЕ [В и В] и в [ВВ]. Тем самым такие конструкции, как

$$a := b + (c := d \times e) - f ,$$

становятся столь же законными, как

<u>вещественная процедура</u> g ; g := c := d × e ;
a := b + g − f

в Алголе 60. В результате любой (в данном смысле) оператор теперь имеет значение и выполнение последовательности n операторов влечет за собой появление на «обрывке бумаги» n значений. Для того, чтобы примирить это

во внимание, разделитель операторов ";", играющий в Алголе 60 чисто синтаксическую роль, становится активной операцией, чье действие состоит в уничтожении значения оператора, выполненного последним. Следовательно блоки, а вместе с ними и собственно процедуры, принимают значения. Например, оператору

начало a:=1; b:=a+1; c:=b+1 **конец**

должно быть присвоено значение 3. Приходится вводить понятие частичной функции, поскольку пустому оператору нельзя приписать никакого значения.

Второе решение, а именно, исключение побочных эффектов, реализуется изменением определения процедуры-функции, состоящим в допущении в качестве тела процедуры выражения, а не оператора (который в Алголе 60 содержит в себе по крайней мере одно особое специальное присваивание идентификатору процедуры). Это решение столь же радикально, что и первое, и приводит к далеко идущим последствиям. В силу концептуальной важности понятия чистого оператора и его роли в способности программы к проверке нет оснований полностью игнорировать этот подход.

2. Проверка и качество программ

Любое построение программы автоматически влечет за собой проверку ее правильности. Тот факт, что в построении программ допускаются ляпсусы (все еще слишком часто), связан с отсутствием систематического (я не говорю "формального") метода проверки. Только совсем недавно была предпринята попытка наметить более четкие принципы разработки такого метода [H], и одна из причин того, что эти принципы широко не применяются,

повторяемых операторов. Этот пример убедительно демонстрирует, как определенные структуры языка и подходящие определения могут обеспечивать методы проверки.

Ниже обсуждаются еще два примера средств, введенных для тех же целей. Оба этих средства базируются не отличном от Алгола во определении операторов цикла, взятом из [В и Х].

3. Границы индексов

Каждый, кого заботит, чтобы неопределенные ситуации фиксировались бы, хотя бы в процессе вычислений — а это должно заботить каждого, кто заинтересован в надежности полученных результатов — должен предусматривать проверку попадания индексов (вычисленных значений) в отведенные им границы. Эта проверка, которая в общем случае может быть проведена только во время выполнения программы, дорого обходится, что делает использование массивов менее привлекательным по сравнению с использованием записей в [В и Х], где обращение к полям не требует никакой проверки, т.к. характер обращения полностью определяется текстом программы. В связи с этим весьма желательно ввести особые обозначения для некоторых общих ситуаций, в которых проверка индексов может быть проведена транслятором. Наиболее подходящим является оператор цикла: если индекс является параметром цикла, то тогда динамическую проверку можно будет опустить, если только транслятор сможет определить, что границы параметра цикла не выходят за границы индекса.

Пример:

<u>вещественный</u> <u>массив</u> a [1:n];

<u>для</u> i := 1 <u>шаг</u> 1 <u>до</u> n <u>цикл</u> S := A[i] + S

Для проверки в этом примере транслятор (и проверяющий человек) должен уметь сравнивать символические величины (в данном случае n) и обнаруживать, что переменной n не производится присваивания между ее описанием и оператором цикла. Эта задача весьма существенно упрощается наличием средства, которое могло бы связывать с именем (идентификатором) некоторые границы изменения.

Пример.

<u>границы</u> R = 1:n
<u>вещественный</u> <u>массив</u> a[R];
<u>для</u> i := R <u>шаг</u> 1 <u>цикл</u> s := A[i] + s

В более общем случае, указание границ изменения могло бы быть связано с описанием переменной, так что каждое присваивание этой переменной сопровождалось бы проверкой на попадание в границы:

<u>целый</u> (R) i

4. Двусмысленные ссылки

Каждое средство, предназначенное для контроля во время трансляции, вводит некоторые ограничения. Важно быть уверенным, что эти ограничения помогают, а не мешают программисту. В этом смысле природа правила, требующего, чтобы каждая ссылка относилась к одному определенному типу (в [B и X] "класс записи"), двусмысленна. Часто желательно, чтобы ссылочные поля в структурах могли бы указывать на структуры разных типов. Эта проблема была рассмотрена в [X], где была предложена весьма разумная концепция подклассов записи, которую здесь можно назвать

ления в отсутствии языков, эти разработчики специально бы рассмотрели эти проблемы. Методы проверки упрощаются, если язык имеет подходящую для этого структуру и если определенные конструкции подчиняются фиксированным правилам проверки.

Под проверкой здесь подразумевается дедуктивное доказательство истинности некоторых фактов о программе, выводимых исключительно из текста программы, т.е. без ее выполнения. Тем самым проверка должна базироваться на информации, которая непосредственно доступна транслятору и которая фактически может быть им использована для автоматического выполнения части проверки. Условия, которые можно будет проверять этим методом, уже не потребуют контроля во время выполнения программы, что сразу повышает качество выполнения. В свете этого можно заключить, что интересы требований качества программ и ее логической ясности совпадают.

Первым примером в свойствах языка, отвечающим этим двум целям, является приписывание фиксированного типа всем переменным в Алголе 60. Ослабление этого правила для формальных параметров то тяжко повлияло на качество выполняемых программ. Аналогичным шагом является привязывание ссылочных переменных к фиксированному классу записи в [ВиХ]. Это содействует как ясности, так и качеству программ настолько существенно, что без этого правила работа с записями была бы вообще малопривлекательной. Другим примером, также из [ВиХ] является оператор цикла, который, в отличие от Алгола 60, допускает, что параметр цикла зависит только от заголовка и не может быть изменен побочными эффектами выполнения

концепцией "категорий" (некоторого определенного "типа"). Определение типа в этом случае принимает форму

тип Лицо (целый возраст; ссыл (Лицо) отец, мать

категория [зачеркнуто] Мужчина (целый номер удостоверения;

ссыл (Лицо) младший ребенок, супруг),

Женщина (логический беременность;

ссыл (Лицо) супруг), Ребенок)

Сначала перечисляются поля, общие для всех категорий, после чего вводятся категории, за каждой из которых следует список (возможно, пустой) "частных" полей.

Ссылка, хранящаяся в ячейке, описанной как

ссыл (Лицо) ч

может теперь указывать на ячейку любой категории Мужчина, Женщина, или Ребенок (которые все по определению принадлежат типу Лицо). В этом случае правильность указателя поля

беременность (ч)

определяется только в момент выполнения. Конечно, программист использует этот указатель поля в предположении (возможно, ошибочном), что ч всегда указывает на Женщину. Обычно он использует предварительную проверку для явного обнаружения этого факта, записывая

если ч это Женщина то ... беременность (ч) ...

Язык должен выражать эту общую ситуацию таким образом, чтобы избежать неявного контроля, связанного с

таким указателем поля. Используемые для этой цели конструкции неизбежно напоминают конструкции, исключающие динамическую проверку границ индексов и использующие понятие величины, которой заведомо не производится присваивание в пределах некоторой области. Нижеследующее обозначение в несколько измененной форме заимствовано у [X]:

```
для t := r когда Мужчина выполнить S1
           когда Женщина выполнить S2
           когда Ребенок выполнить S3
```

Здесь r – ссылочное выражение, S1, S2, S3 – операторы, а t – локальная величина, описываемая неявно аналогично предыдущему примеру.

Многорегистровые машины могут эффективно использовать преимущества такой конструкции, засылая часто используемую в операторах S1, S2 и S3 величину t в регистр.

5. Процедуры как элементы данных

В Алголе процедуры являются статическими по природе и не могут быть данными. Процедуры нельзя вырабатывать, их можно только выполнять. Эффект вызова процедуры "издалека" путем оператора процедуры или изменения процедуры, являющегося параметром, описывается с помощью текстуальной подстановки (правило копирования).

В ЭЙЛЕРЕ было реализовано понятие вырабатываемой процедуры разрешением упоминать процедуры косвенно с помощью ссылок, которые могут, в свою очередь, присваиваться переменным. Это решение наиболее привлекательным образом объединяет эти две концепции Алгола 60 — параметр-процедура и вызов именем. Обозначение буквальной процедуры имеет вид текста процедуры, взятого в кавычки и поэтому называемого выдержкой. Для того, чтобы выразить эту ситуацию в языке типа Алгола, должны быть введены элементарные значения типа <u>процедура</u>, с которыми можно работать следующим образом:

```
A:      процедура p;
B:      p := 'x := x+1' ;
C:      p
```

Вот смысл этого куска текста: в A вводится переменная (ячейка) p, в B ячейке p присваивается выдержка, а в C вхождение p означает выполнение этой выдержки. Обозначение, более согласованное с предыдущими разделами, имеет следующий вид:

```
A:      ссыл [Процедура] p, q ;
B:      p := Процедура(x := x+1);
C:      вып p ;
D:      q := p
```

Это обозначение позволяет присваивать переменным подходящего типа ссылки на процедуры без выполнения последних.

Как и для ~~[illegible]~~ [illegible] в предыдущих разделах, концепция присваивания текста процедуры в сочетании с блочной структурой, к несчастью, ведет к неисчислимым трудностям. Здесь эти они даже еще более чувствительны, поскольку с очевидностью текст программы может содержать имена (идентификаторы) с ограниченными областями действия и в то же время присваиваться величинам, находящимся вне этих областей.

Пример:

```
начало процедура p; целый i, k;
    i := 100;
    начало целый i, j;
        p := 'k:=i+j';
        i := j := 10; p
    конец
    p
конец
```

Первое выполняемое вхождение p вполне законно, в то время как второе в высшей степени проблематично.

6. Циклы

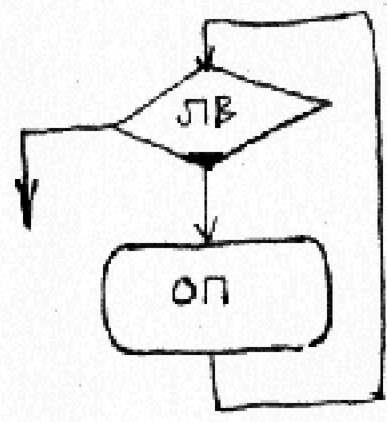
Цикл — это в высшей степени часто встречающаяся программная конструкция, для которой должны существовать простые и к тому же гибкие обозначения. Программисты, которые привыкли описывать свои алгоритмы в терминах блок-схем, при их трансляции в последовательные языки склонны часто использовать операторы перехода. Здесь предполагается, что циклы должны в первую очередь встречаться в виде прямых программных структур, нежели

путем утомительного использования меток, которые не только ведут к перерасходу идентификаторов но и которые весьма способствуют превращению программ в запутанный лабиринт путей.

В [Bu X] была предложена простая конструкция для простого цикла в виде:

пока <логическое выражение> цикл <оператор>

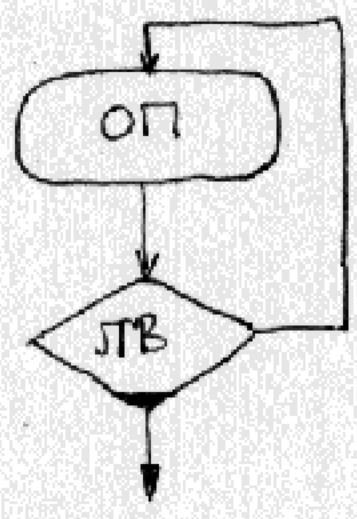
изображаемая блок-схемой в виде



Я думаю, что эта конструкция имеет преимущества перед Алголовским оператором цикла с элементом заголовка типа "пока" - благодаря своей простоте и очевидности. Однако эта конструкция иногда недостаточна, потому что она требует обязательной проверки условия еще до первого выполнения оператора. Бэрроузский Расширенный Алгол (для Б 5500) компенсировал этот недостаток, предложив аналогичную конструкцию, предписывающую выполнение оператора перед проверкой на окончание:

цикл <оператор> до <логическое выражение>

72-222



Кажется, однако, что это только частичная компенсация. Фактически, часто появляются куски программ следующего вида

S1; <u>пока</u> ЛВ <u>цикл</u> <u>начало</u> S2; S3 <u>конец</u> <u>на</u> L;

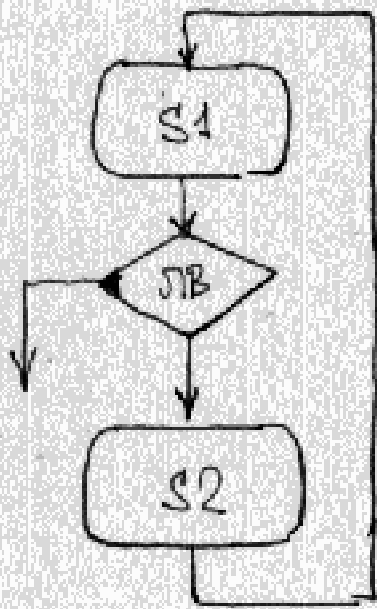
<u>цикл</u> <u>начало</u> S2; L: S3 <u>конец</u> <u>до</u> ЛВ

где S1 и S3 – одинаковые последовательности операторов. Топологическое подобие этих конструкций подсказывает необходимость собственной конструкции*) состоящей из двух операторов и проверки на окончание цикла, выделенных подходящими ограничителями. Вот что предлагается здесь:

<u>повторить</u> <u>начало</u>
S1;
<u>когда</u> ЛВ <u>выход</u>;
S2
<u>конец</u>

---

*) Идея была высказана Доном Кнутом в связи с ПЛ/1 для 360. Это также фигурирует в виде оператора ВЫЙТИ ИЗ в Процедуре Потока для Б 5500



Ранее рассмотренные операторы "пока" и "до" являются операторами повторения с, соответственно, пустыми S2 или S1.

Возможен другой выбор ограничителей:

<u>приступить</u> <u>начало</u>
S1;
<u>когда</u> ЛВ <u>пропустить</u>
S2
<u>конец</u>

72-224

Литература

[Х] C.A.R. Hoare, "Record Handling", lectures given at Nato Summer School on Programming, 1966.

[Н] P. Naur, "Proof of Algorithms by General Snapshots", BIT 6, 4 (1966) pp. 310-316.

[Р] D.T. Ross, "AED Language", Electronic Systems Lab., MIT.

[вВ] A. van Wijngaarden, "Proposal for a Successor to Algol", Working Document "Warsaw 2", IFIP WG 2.1

[В и Х] N. Wirth and C.A.R. Hoare, "A Contribution to the Development of ALGOL", Comm. ACM, 9, 6 (June 1966).

[В и В] N. Wirth and H. Weber, "EULER, A Generalization of ALGOL", Comm. ACM 9, 1-2 (Jan./Feb. 1966).