Школа современного программирования

Романовский Иосиф Владимирович

ЗАМЕЧАНИЯ О СТИЛЕ ПРОГРАММИРОВАНИЯ

Этот вопрос всегда был интересен для меня: я считал, что программы пишутся не только для транслятора, но и для человека. В 1960-е годы мы издавали в ВЦ Ленинградского университета «Алгол-процедуры» – сборники написанных в алголе 60 программ. Одной из целей была именно выработка хорошего стиля. Несколько лет назад я редактировал перевод книги Б. Кернигана и Р. Пайка «Практика программирования» [1], и для меня было большой радостью, что позиции авторов по многим вопросам очень близки к моим.

Но из собственного преподавания постепенно эти вопросы уходили. Правда, я читаю на кафедре исследования операций в 5 семестре спецкурс, включающий и эти вопросы, но эффективность его не была высокой. Летом 2004 г. я сделал ошибку, поставив пятерку за дипломную работу, в которой попиралось все, чему автора работы учили на матмехе в течение всех пяти лет. Все лето я сам себя воспитывал и истязал, стараясь последовательным переписыванием довести программное творение автора до нужных кондиций. До сих пор совершенство еще не достигнуто.

Но уже хочется поделиться накопленным богатством с другими. Итак, на какие вопросы следует обратить внимание. (При обсуждении этих вопросов один из собеседников заметил мне, что я говорю здесь просто о стандартных требованиях к программированию. Это правда. Но до тех пор, пока приходится эти стандартные требования напоминать, давайте это делать.)

Не следует писать русские слова латинскими буквами...

1. ИДЕНТИФИКАТОРЫ

а) **Не следует писать русские слова латинскими буквами**, а тем более делать латинские формы от русских слов и писать как бы по-английски русифицированное английское слово.* Самые кошмарные примеры последнего рода – это `fail` для обозначения файла и `stek` для стека.

Из русских слов на меня самое большое впечатление в упомянутом тексте произвели булева переменная `gotovo`, которую все время хотелось принять за оператор перехода, массив `vershinas` и серия булевских переменных с названиями типа `daleeq`.

* Особенно трудно выполнить это требование, когда (скажу вам по секрету) prepodavatel' sam trebuet, chtoby identifikatory byli russkimi i pisalis' v latinitse.

...нужно преследовать студента за произвольную смену регистров букв.

Кроме таких, почти русских, названий были и помеси, вроде `ShowOstTree`. Нужно пояснить, что в теории экстремальных задач на графах используется термин «остовное дерево», соответствующий английскому «spanning tree». Химера `SdvigChild`, по-видимому, ясна без пояснений.

Про такие сравнительно невинные названия как `obraz`, `hvatit`, `narod`, `krazr` и т. п. я уж и не говорю.

б) **Не следует писать одно и то же английское слово в разных вариантах** (конечно же, из всех вариантов нужно стараться выбирать правильный). Совсем недавно я исправлял слово `hesh` на `hash`. Опасность в том, что автор потом может забыть о том, что он отличается от прочего мира, более грамотного, и, используя введенное где-то далеко слово, может его случайно написать уже правильно. Кстати, это может быть связано и с использованием английских написаний слов вместо американских (например, `Colour` вместо `Color`).

в) **Нужно выбирать размер идентификатора в зависимости от частоты его появления** (чем чаще, тем короче, конечно). В иностранных книжках по хорошему стилю в программировании я встречал «образцы понятных идентификаторов» примерно такого типа:

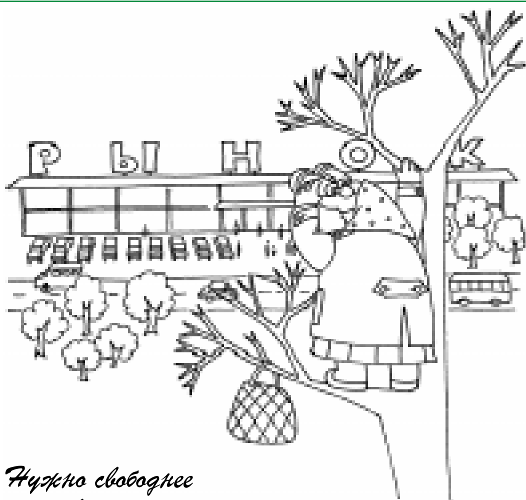
```
basicmatrix[rowindex,columnindex] := basicmatrix[rowindex,columnindex]*value[columnindex];
```

Вместе с тем, я бы не вводил, как иногда предлагается, строгих априорных ограничений идентификаторов по длине. Почему? Потому что автор должен думать не о соблюдении внешнего ограничения (идентификаторы `i32`, `i33`, `i97` и т. п. очень лаконичны), а о понятности текста.*

г) При использовании систем визуального программирования (типа Visual C или Delphi) **не следует пользоваться идентификаторами для убогих – автоматически создаваемыми именами компонентов, установленных на форму.** Все названия типа `Button1`, `Button2`, `Label18` нужно сразу же, не ленясь, заменять осмысленными. Это же относится и к названиям модулей: `Unit1` и т. д. Ужасные последствия лени ощущаются после того, как человек уже расставил несколько десятков таких элементов и тратит время на размышления, что дороже: продолжать в том же духе или переименовывать (безусловно, с возможностью новых ошибок).

д) В Паскале и родственных языках **нужно преследовать студента за произвольную смену регистров букв.** Очень плохо, если одно и то же слово в соседних строчках написано по-разному: `StrToInt`, `Strtoint`, `strtoint`. Даже такие простые слова как `True` и `False` не должны смешиваться с `true` и `false`. С моей точки зрения омерзительны провоцируемые использованием Word написания служебных слов с большой буквы. Почему омерзительны? Потому что эти фокусы крадут мое читательское время – я невольно начинаю размышлять «А зачем он (она) здесь поставил(а) большую букву?». Убежден, что вы сейчас задумались, зачем Романовский написал эту фразу в гендерно-корректном виде (то есть не обижающем представителей никакого пола). Я-то это сделал для неожиданного «подарка» читателю: почувствуй-

* В. О. Сафонов рассказал мне о замечательном прецеденте: В одной организации настоятельно рекомендовалось делать глобальные идентификаторы пяти-символьными, а локальные трех-символьными.


Нужно свободнее пользоваться пробелами.

те, как такие неожиданности мешают пониманию.

Нарушение этого естественного правила я встретил даже в книгах опытных педагогов.

Мои привычки здесь таковы: я пишу строчными (маленькими) буквами все служебные слова, кроме `Boolean`, `True` и `False`. Системные процедуры я пишу так, как это рекомендует система: `Inc`, `Dec`, `ShowMessage`, `ShowMessageFmt` и т. п.

е) Всяческого распространения заслуживает так называемая **венгерская система обозначений**, в которой идентификатор начинается с написанного строчными буквами префикса, определяющего тип объекта (например, `s` для `string`, `r` для `real`, `f` для `file`, и соответственно для компонентов формы: `bt` для `Button`, `sg` для `StringGrid` и т. п.).

Далее индивидуальная часть идентификатора обязательно начинается с большой буквы.

Отмечу, что, пропагандируя это правило, сам я регулярно «ловлюсь» на том, что использую для буферной строки вводимых данных идентификатор `sIn`, который компилятор естественно отождествляет со стандартной функцией.

ж) **Обозначения для констант.** Очень четко этот вопрос ставится в [1]: в правильно написанной программе числовые константы в явном виде почти не должны встречаться.

Они должны быть заданы посредством именованных констант, которые и должны затем использоваться.

Даже, если вам нужно использовать количество дней в неделе, то нельзя писать просто 7, а нужно сделать описание вроде

```
const nWeekDays = 7;
```

и далее в тексте употреблять этот идентификатор (чтобы отличить от возможных появлений того же числа в другом смысле, например, для выбора номера месяца – июля).

Совсем уж безобразно выглядит буквальное воспроизведение системных констант, например, для установки белого цвета запись

```
pen.color:=$00FFFFFF;
```

вместо

```
pen.color := clWhite;
```

В упоминаемом мною тексте автор несколько десятков раз использует радиус в пикселях изображаемой на картинке точки. Он выбрал значение 5. Представьте себе, что будет, если он или его научный руководитель решит попробовать другой радиус. На самом деле, вводя процедуру рисования точки, он уменьшил бы число вхождений до четырех, но и в этом случае правильно было бы написать

```
const kR = 5; // радиус рисуемой точки
```

2. РАСПОЛОЖЕНИЕ

а) **Пробелы в строке.** Нужно свободнее пользоваться пробелами. Знак присваивания и знаки операций должны, как говорят полиграфисты, «отбиваться» от окружающего текста.

В качестве примера рассмотрим, несколько забегая вперед, фрагмент студенческой программы (пока маленький кусочек)

```
Brush.Color:=clWhite;
TextOut(0,15,copy(s,1,i-1));x1:=PenPos.X;
if l=1 then begin
Brush.Color:=$00E6E6E6;
TextOut(x1,15,' '+s[i]+' ');x1:=PenPos.X; end;
Brush.Color:=$00FEE6C5;
TextOut(x1,15,copy(s,i+1,m-1));
```

Ясно, что и на строки делить нужно не так и отступы нужно делать.

```
Brush.Color := clWhite;
TextOut(0,kTop,copy(s,1,i-1));
x1 := PenPos.X;
if l=1 then begin
  Brush.Color := clA;
  TextOut(x1,kTop,' '+s[i]+' ');
  x1 := PenPos.X;
end;
Brush.Color := clB;
TextOut(x1,kTop,copy(s,i+1,m-1));
```

Я позволил себе ввести обозначения для цветов и для числа 15, предполагая, что это вертикальная позиция выводимой строки.

Меня очень раздражают расстановки пробелов в записях вида

```
if( k=5 )then
```

Кто-то из студентов сказал мне, что такая запись оправдывается встроенными редакторами сред программирования, которые учитывают синтаксис:

```
if( k=5 )then
```

Но все равно с правильной расстановкой пробелов текст нагляднее. Сравните:

```
if (k = 5) then
```

б) **Отступы.** Я запрещаю себе использовать знак табуляции (он в разной среде интерпретируется по-разному и это неудобно), и для меня почти всегда отступ означает две позиции, как это показано в приведенном выше примере. По понятным резонам – одной позиции мало, а больше жалко. Почему жалко? Потому что в сложной программе вложенность уровней может быть довольно высокой (больше десятка – это еще не предел).

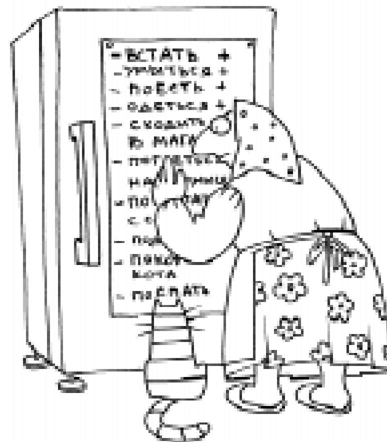
в) **Расположение операторных скобок.** Обычно рекомендуется каждому программисту выработать свой собственный стиль. Я придерживался этого либерального подхода, но сейчас частично изменил ему и требую, чтобы **в программах для меня операторные скобки располагались так, чтобы мне было удобно.** Теперь мне хотелось бы убедить коллег, что им нужно требовать у студентов того же самого и не устраивать разнобоя. Керниган и Пайк рекомендуют ставить `begin` в той же строке, что и использующая его конструкция (например, цикл, условие или `with`), а end выравнивать по началу этой конструкции (см. выше условный оператор). Такое расположение хорошо подчеркивает структуру программы и не провоцирует лишних строчек и лишних сдвигов (как это бывает при расположении `begin` в отдельной строчке). Рассуждения о парности `begin` и `end` не вполне честны, `end`, как известно, не всегда стоит в паре с `begin`.

г) **Принцип «один оператор в строке».** По-моему, в использовании этого принципа нужна умеренность. В некоторых случаях, напротив, нужно объединять в одной строчке операторы, собирающие более крупное действие. Например, если мы манипулируем с трехмерными геометрическими объектами и почему-либо храним координаты точек не в единой структуре, а по отдельности, «присваивание точки» состоит из трех покоординатных присваиваний, которые правильнее записывать в одной строке, вроде следующего

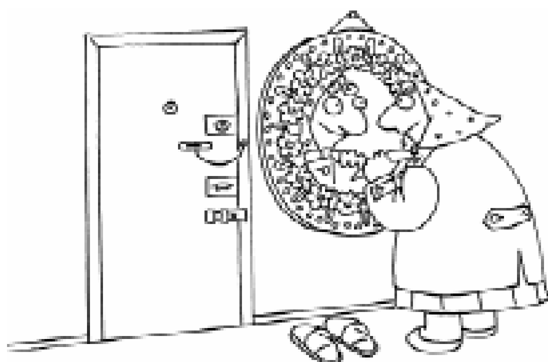
```
xCur := x[i]; yCur := y[i]; zCur := z[i];
```

При рисовании отрезка на «холсте» в системе Дельфи, операторы MoveTo и Lineto, по-моему, также правильнее располагать в одной строке:

```
MoveTo(x1,y1); LineTo(x2,y2);
```



Принцип «один оператор в строке».



Все эти украшения сильно усложняют чтение самой программы...

Существенно улучшается от такой группировки «читабельность» текста, который состоит из чередующихся операторов, скажем типа `A` и `B`, вроде:

```
A(P1); B(P1); A(P2); B(P2); A(P3);
B(P3); A(P4); B(P4);
```

Не могу себе позволить разместить их все в один столбик, но частично должен.

```
A(P1);
B(P1);
A(P2);
B(P2);
A(P3);
B(P3);
```

Это не только неэкономно, но и раздражает чередованием операторов. Расположение их парами с нарочитым выравниванием упрощает в этом случае текст очень существенно.

```
A(PFirst); B(PFirst);
A(PSecond); B(PSecond);
A(PThird); B(PThird);
A(PFourth); B(PFourth);
```

Некрасиво! Так лучше:

```
A(PFirst);  B(PFirst);
A(PSecond); B(PSecond);
A(PThird);  B(PThird);
A(PFourth); B(PFourth);
```

д) **Изменение расположения в ходе работы над программой.** Совершенно не обязательно сохранять одно и то же расположение в течение всей работы над программой. Хорошо проверенные фрагменты можно и нужно писать несколько компактнее (однако, не усердствуя), а фрагмент, в котором разыскивается ошибка можно временно «распотрошить» на мелкие строчки, снабдив их подробным комментарием. К оным мы сейчас и обратимся.

3. КОММЕНТАРИИ

Заставить писать комментарии очень трудно, я не могу до конца заставить и самого себя.

Кроме того, меня раздражает распространенный стиль обширных комментариев с дополнительным выделением «заголовков» из звездочек, знаков равенства, косых палочек и т. п. Все эти украшения сильно усложняют чтение самой программы (как современные изменяющиеся рекламы на футбольном поле все больше мешают смотреть сам футбольный матч).

Но можно выделить несколько типов комментариев, на которые следует обратить внимание.

а) **При описаниях типов, констант и переменных.** Здесь достаточно писать всего по несколько слов. Лучше, если комментарий будет помещаться в той же строке, что и описываемый идентификатор, и комментарии будут выровнены. Допустимы отклонения при комментировании «из ряда вон выходящих», то есть более длинных, чем остальные, идентификаторов и идентификаторов, по какой-либо причине исключительных. Такое отклонение может быть временным.

б) **При разметке структуры текста.** Это комментарии всего в одну строку. Как заголовки в книгах. Вспомним де Голля, в мемуарах которого название каждой главы состояло из одного слова.

К этой же категории я бы отнес комментарии к ветвям условного оператора. Их хорошо размещать (если получается) после открывающей операторной скобки.

в) **После закрывающей операторной скобки.** Достаточно совсем коротких комментариев, помогающих в идентификации многих завершений. Например:

```
      end; {for j}
    end; {while}
  end; {case}
end; {for i}
```

Эти комментарии можно добавлять по мере надобности в тех местах, которые вызывают трудности.

г) **Перед процедурами (функциями).** Не требуется писать их всегда, и уровень описания может быть различным.

В некоторых случаях можно не писать ничего.

В других – описывать действие, выполняемое процедурой (смысл параметров, иногда используемый метод).

В третьих – нужна полная спецификация всех параметров, результата и побочных эффектов.

В четвертых – подробная детализация используемых данных, возможно с примером.

Если описание получается очень сложным, то полезно разбить его на две части: перед самой процедурой оставить минимальный комментарий, а длинное объяснение перенести в конец файла.

д) **В конце файла.** Здесь можно себе позволить все, что угодно. Целый научный трактат о методе и, если хочется, образец отладочных данных с ожидаемым результатом. Очень полезно в долгоживущих программах иметь в конце историю изменений. Иногда ее пишут в обратном хронологическом порядке, это очень удобно для читателей с ограниченным ресурсом времени и памяти.

е) **В начале файла.** Вот комментарий для нас совсем непривычный и почти обязательный на Западе. В каждом файле должна в начале содержаться «учетная информация». Обязательно должно быть написано имя самого файла. Затем имя автора, время изготовления, название проекта и назначение данного файла. Если угодно, запись о копирайте.

Например:

```
// da_02_11_lexperm.pas
// Процедура лексикографического
// перебора перестановок
// И. В. Романовский, 2004-10-29
// Пример заголовка файла.
// Самой такой программы нет
```

Возможно, что потом будет.

ж) **Грамматические ошибки в комментариях.** Я пришел в бешенство, увидев несколько раз в комментарии слово **вершына** (именно так). Понятно, что обычная общечеловеческая грамотность не влияет формально на правильность исполнения программы*. Но это формально. А на самом деле этим автор демонстрирует свое отношение к работе. Думаю, что здесь можно пойти на публикацию подобных случаев, и с упомянутым комментарием я так и поступил.

4. РЕКОМЕНДУЕМЫЕ И НЕ РЕКОМЕНДУЕМЫЕ КОНСТРУКЦИИ

Керниган и Пайк обращают особое внимание на использование стандартных средств используемого языка программирования.

а) Из таких средств я бы в первую очередь назвал **структурные типы данных**. Привитие навыков описания структурных данных очень важно: они совершенно необходимы при работе, например, с базами данных. Нужно объяснять, чем удобны структурные типы для компьютеров, в частности, как выборка полей структуры поддерживается форматом машинной команды (смещение для данного поля записывается прямо в команде). Важно разъяснять проблемы, возникающие в связи с так называемым выравниванием.

Нужно обращать внимание на удобства использования замечательной конструкции `with` из языка Паскаль. Если у нас описан, например, массив структур `aggregate` и мы работаем с каким-то конкретным элементом этого массива, то можно написать так

```
var agA: array[1..40] of aggregate;
. . . . . . . . .
for i := 1 to 40 do with agA[i] do begin
  iA := 0;   iB := i;
  cD := 'n'; rX := 0.0;
end;
```

* Недавно в спаме пришло письмо, в котором Subject был: Не дают прохода ковалеры? Но еще Пушкин писал: «Как уст румяных без улыбки,/Без грамматической ошибки/Я русской речи не люблю...»

«Детский» стиль ввода.

Использование with, с одной стороны, подсказывает компилятору, что предстоит несколько действий с одним и тем же структурным набором agA[i], с другой стороны, облегчает запись программисту.

Сравните с тем же текстом без with

```
for i := 1 to 40 do begin
   agA[i].iA := 0;   agA[i].iB := i;
   agA[i].cD := 'n'; agA[i].rX := 0.0;
end;
```

б) Я стараюсь предупреждать студентов о нежелательности использования **многомерных массивов**. Мне кажется, что трех- и более мерных массивов быть в практике программирования почти не должно. Часто еще встречается принудительное заталкивание данных переменного размера с непрямоугольным множеством индексов в обычный двумерный массив (скрытое выравнивание).

в) **«Детский» стиль ввода.** Со школьных времен у студентов остается (и сохраняется даже до третьего курса) привычка к вводу параметров в диалоговом режиме. Хороший способ с этой привычкой бороться – предложить студенту ввести матрицу, например, 10 на 10, отменить ввод, а затем попросить ввести ее снова, и так еще 2–3 раза. Но этим дело не кончается.

Согласившись на ввод из файла, они вписывают в программу полное имя этого файла с путем, настроенным на свою домашнюю (или университетскую) машину, совершенно не заботясь о том, как машина найдет указанный ими путь на преподавательской машине. В прошлом семестре я «воевал» с третьекурсником, который писал так:

```
assign(dataD,'w:\min\data\d.txt');
reset(dataD);
assign(dataW,'w:\min\data\w.txt');
reset(dataW);
```

После моих разъяснений он ничего не изменил. Пришлось объявить бойкот до получения от него исправленного варианта.*

Попутно обнаружилось, что пользоваться условными путями типа .\data или ..\input студенты не умеют.

Знания студентами передачи через командную строку параметров запуска программы я не встречал ни разу. Интересно, что студентов за границей учат, прежде всего, вводу, выводу и контакту программы с операционной системой. При разговоре со студентами третьего курса формат обращения к параметрам запуска (функция ParamCount выдает число параметров, а функция ParamStr(i) – параметр i) знал только один человек, опыта работы с этими параметрами не имел никто, а несколько человек не имели никакого представления о командной строке.

Я при первой возможности спросил о командной строке первокурсников. В принципе о ней были осведомлены все, но приведенная для контроля строка

```
dir p*.tex > list_tex
```

уже вызвала вопросы человек у десяти и запись объяснения много большим числом студентов.

Потребность задания имени файла настолько не осознается, что даже студенты, использующие продвинутую визуальную систему (Дельфи и т. п.), компонента OpenDialog не знают (может быть, я должен объяснить и читателю, что речь идет о стандартном окне для выбора имени файла?).

г) **Использование операций приращения для целочисленных переменных.** В

---

* Сам по себе бойкот не помог. Я вызвал его к доске на семинаре, и целый час мы разбирали на семинаре ввод и разбор параметров командной строки. Неожиданно понадобилось рассказывать и про функции Pos и Copy.

Паскале есть удобные функции `Inc` и `Dec`. Они пишутся короче, чем обычные операторы присваивания, особенно при сложном задании изменяемой переменной. Например, лучше писать `Inc(a[iR,iC])`, чем `a[iR,iC] := a[iR,iC]+1`.

д) **Стандартные компоненты.** Мне кажется, что пора уже учить даже первокурсников использованию стандартных компонентов, обеспечивающих пользовательский интерфейс. Дельфи в этом отношении очень просто обеспечивает весь необходимый минимум (это сказано очень скромно – обеспечивается все нужное).

д) **Форматный вывод.** Когда смотришь американский учебник программирования, то сначала создается впечатление, что главное в программировании – это красивый вывод результата. А мы, вроде бы, выше этого. И вот наши студенты пишут (пример реальный и типичный) (листинг 1).

Обратите внимание, это уже далеко продвинувшийся студент, которому велели красиво вывести данные большого эксперимента. Он уже умеет форматировать в Паскале числовые поля. Но если бы он знал функцию Format, введенную в Дельфи по примеру Си, он мог бы записать этот фрагмент так (см. листинг 2) или даже так (см. листинг 3), а то и так (см. листинг 4).

Отдельное определение форматных строк для вывода следует, по-моему, очень поддерживать.

Листинг 1

```
writeln(f,'Оценка 0 дала лучший результат в '+inttostr(bests[3])+' случаях.
('+floattostrF(bests[3]/kmaps*100,ffGeneral,8,5)+'%)');
writeln(f,'Оценка 1 дала лучший результат в '+inttostr(bests[4])+' случаях.
('+floattostrF(bests[4]/kmaps*100,ffGeneral,8,5)+'%)');
writeln(f,'Оценка 2 дала лучший результат в '+inttostr(bests[5])+' случаях.
('+floattostrF(bests[5]/kmaps*100,ffGeneral,8,5)+'%)');
writeln(f,'Оценка 3 дала лучший результат в '+inttostr(bests[6])+' случаях.
('+floattostrF(bests[6]/kmaps*100,ffGeneral,8,5)+'%)');
```

Листинг 2

```
writeln(f,Format('Оценка 0 дала лучший результат в %d случаях.(%8.5f%)',
  [bests[3],bests[3]/kmaps*100]));
writeln(f,Format('Оценка 1 дала лучший результат в %d случаях.(%8.5f%)',
  [bests[4],bests[4]/kmaps*100]));
writeln(f,Format('Оценка 2 дала лучший результат в %d случаях.(%8.5f%)',
  [bests[5],bests[5]/kmaps*100]));
writeln(f,Format('Оценка 3 дала лучший результат в %d случаях.(%8.5f%)',
  [bests[6],bests[6]/kmaps*100]));
```

Листинг 3

```
sFormatStat := 'Оценка %d дала лучший результат в %d случаях.(%8.5f%)';
writeln(f,sFormatStat,[0,bests[3],bests[3]/kmaps*100]));
writeln(f,sFormatStat,[1,bests[4],bests[4]/kmaps*100]));
writeln(f,sFormatStat,[2,bests[5],bests[5]/kmaps*100]));
writeln(f,sFormatStat,[3,bests[6],bests[6]/kmaps*100]));
```

Листинг 4

```
sFormatStat := 'Оценка %d дала лучший результат в %d случаях.(%8.5f%)';
for kStat := 0 to 3 do
  writeln(f,sFormatStat,[kStat,bests[kStat+3],bests[kStat+3]/kmaps*100]));
```

...главное в программировании — это красивый вывод результата.

5. ОТЛАДКА

Я так и не научился пользоваться отладочными средствами современных систем программирования и использовал отладочные печати, которые легко включал и выключал, иногда довольно хитрым образом. В некоторых случаях, занимаясь геометрическими расчетами, я даже выводил для отладки результаты в форме постскриптовских файлов, чтобы рассмотреть промежуточную картинку. Мне было очень приятно узнать, что Керниган и Пайк рекомендуют не увлекаться стандартными отладочными средствами. Они объяснили мне то, что я только чувствовал.

а) **Преимущества отладочных печатей**. Просто перечислю некоторые из преимуществ.

Прежде всего, использование отладочных печатей **более универсально**, чем встроенная система отладки и легко переносится на другую систему.

Я так и не научился пользоваться отладочными средствами.

Далее, **возможности форматирования вывода при использовании отладочных печатей шире, чем при системной отладке** – некоторые параметры можно формировать «на лету», не заботясь о хранении данных в какой-либо переменной.

Отладочные печати можно сопровождать **программно проверяемыми условиями вывода.**

В отличие от чисто диалоговых системных средств, отладочные печати **можно сохранять в тексте программы**. Формы сохранения различны: их можно прятать в комментариях, а можно защищать условными операторами, зависящими от ключей отладки. Это обстоятельство позволяет возвращаться к приготовленным ранее средствам анализа поведения программы.

Очень важна также **возможность формирования файла** отладочных печатей с данными для последующего поиска в нем нужного места или обработки другими программными средствами. Я неоднократно сам пользовался этим, передавая данные в `Excel` или, как отмечалось выше, в `PostScript`.

б) **Внимание к сообщениям транслятора.** Предупреждения транслятора обычно игнорируются. Я помню одного системного программиста в нашем Вычислительном Центре, который пользовался возможностями транслятора на ЕС и отключал все системные выдачи кроме «серьезных ошибок». На первый взгляд у него был резон: ошибок было очень много, даже из категории «серьезных». К сожалению, никто был не в силах объяснить этому системщику, что большинство его «серьезных» ошибок было наведено ошибками, которые упоминались даже не в обычных ошибках, а в предупреждениях.

Нужно добиваться того, чтобы не было даже предупреждений. Действительно, если в вашей программе обычно есть более десятка предупреждений, то появление нового не будет замечено, а в нем могут быть указаны опасные изменения в программе.

6. ЗАЧЕМ НУЖНО ДОБИВАТЬСЯ ПРАВИЛЬНОГО СТИЛЯ

Программы писать трудно. Большие программы писать очень трудно, и даже корифеи программирования не могут избавиться от ошибок.

По-моему, **одна из самых больших проблем для автора большой программы – это правильное распределение своего** (увы, ограниченного) **внимания между отдельными частями программы.** Фактически все, о чем говорилось в этих заметках, это вопросы рационального распределения внимания.

Литература

1. *Керниган Б., Пайк Р.* Практика программирования, СПб., «Невский диалект», 2001.

Романовский Иосиф Владимирович, доктор физико-математических наук, профессор СПбГУ.

Наши авторы, 2005.
Our authors, 2005.