УДК 004.4'23

Автоматизация контроля стиля учебных программ

Д. А. Васенина[1], М. А. Плаксин[2]
[1]Пермский государственный университет, Россия, 614990, Пермь, ул. Букирева, 15
natsume_maia@mail.ru; (342) 2-396-594
[2]Пермский государственный университет, Россия, 614990, Пермь, ул. Букирева, 15
Государственный университет «Высшая школа экономики» (Пермский филиал),
Россия, 614070, Пермь, ул. Студенческая, 38
mapl@list.ru; (342) 2-396-594

Описывается программа для автоматизации контроля хорошего стиля учебных программ на языке Паскаль. Программа позволяет автоматизировать часть работы преподавателя по обучению основам программирования.

Ключевые слова: *программирование для ЭВ; стиль программирования; автоматизация обучения.*

1. Введение

Описанная в данной статье программа StyleChecker является частью комплекса средств по автоматизации труда преподавателя, ведущего начальное обучение программированию. Она предназначена для автоматизации проверки стиля программ начинающих программистов.

Обучение программированию включает в себя ряд аспектов: постановку задачи, содержательное решение задачи, запись программы на выбранном языке программирования, правильное оформление, тестирование, отладку и др. Причем учить всему этому приходится одновременно, в комплексе. В результате внимание рассредоточивается сразу по нескольким направлениям. Это требует значительных усилий как от обучаемого, так и от преподавателя.

В настоящее время проблема обостряется в связи с двумя факторами.

Во-первых, в вузах расширяется число специальностей, при поступлении на которые придется сдавать экзамен по информатике. А поскольку основной формой сдачи вступительных экзаменов является ЕГЭ, соответственно возрастет число выпускников, сдающих ЕГЭ по информатике. Самая сложная (и дорогая по баллам) часть ЕГЭ – часть С. Она включает в себя написание программ. Соответственно школы будут вынуждены уделять больше внимания подготовке к экзамену по информатике вообще и обучению программированию, в частности. То есть количество школьников, которые начнут обучаться азам программирования, в ближайшее время возрастет.

Во-вторых, в вузах в связи Болонским процессом идет сокращение аудиторных часов и увеличение часов на самостоятельную работу студентов.

Для того чтобы уменьшить нагрузку на преподавателя, предлагается автоматизировать его работу. Создание единой программы "автоматического преподавания информатики" представляется слишком сложным. Более перспективным кажется разделение обучения программированию по нескольким направлениям и автоматизация преподавания по каждому из них отдельно.

Данная работа посвящена автоматизации проверки стиля учебных программ.

© Д. А. Васенина, М. А. Плаксин, 2010

2. Постановка задачи автоматизации контроля стиля учебных программ

Под стилем понимается определенный набор правил записи текста программы.

Правильное оформление программы способно весьма существенно упростить ее понимание. И наоборот, небрежность в оформлении может сделать даже простую программу совершенно непонятной.

На рис.1 приведен пример плохо оформленной программы. Содержательного смысла в ней нет, она просто демонстрирует пример плохого стиля.

```
program a1;
var oo,o0:real;ll,il:string; begin read(oo);o0:=oo+2;
readln(ll;readln(il);if o0>10 then writeln(ll,o0)) else wri-
teln(il,o0);end;
```

Рис. 1. *Пример плохо оформленной программы*

В таком нагромождении слов очень сложно понять, что делает программа. Еще сложнее найти в ней ошибку.

Имена переменных не несут никакой смысловой нагрузки, близки по написанию и звучанию. Расположение текста никак не связано со структурой программы, никак не способствует выделению составляющих программу языковых конструкций.

Для сравнения на рис.2 приводится та же программа, но записанная в более читабельном виде.

```
program test;
const gran = 10;          {граница}
var num,                  {вводимое число}
    res: real;            {получаемое число}
    DiagUslVyp,           {диагностические сообщения}
    DiagUslNeVyp: string;
begin
   writeln('Введите, пожалуйста, исходную величину');
   read(num);
   res:=num+2;
   writeln('Что делать, если условие выполнено?');
   readln(DiagUslVyp);
   writeln('Что делать, если условие НЕ выполнено?');
   readln(DiagUslNep);
   if num > gran then
      writeln(DiagUslVyp,':',Res)
   else writeln(DiagUslNeVyp,':',Res);
end;
```

Рис. 2. *Пример хорошо оформленной программы*

В данном случае для повышения читабельности программы используются такие нехитрые приемы, как мнемонические идентификаторы, абзацные отступы, комментарии, именованные константы.

Подразумевается, что StyleChecker "знает" некоторый набор правил оформления программ, способен проверить, соблюдаются ли эти правила при записи конкретной программы, выявить отклонения, место нарушения и указать, в чем именно нарушение состоит.

Поскольку в любых правилах могут быть исключения, сообщения StyleChecker’a рассматриваются не как обязательные требования, а как рекомендации. Однако отступление от этих рекомендаций должно быть обосновано.

Планируется, что студент будет сначала проверять свою программу StyleChecker’ом. И только после его "одобрения" представлять ее на проверку преподавателю.

StyleChecker является аналогом "взрослых" программ для проверки стиля программного текста, таких как Microsoft.StyleCop, FxCop и др. Главные его отли-

чия: ориентация на язык Паскаль и на простые программы студентов, которые только начинают изучать программирование. Профессиональные "проверяльщики" ориентированы, как правило, на C#, встроены в профессиональные системы программирования (типа Visual Studio) и представляются слишком сложными для начинающих.

3. Противоречия при обучении хорошему стилю программирования

Обучение новичков хорошему стилю программирования связано с рядом противоречий.

Главное из них заключается в том, что будущих профессиональных программистов надо учить писать большие программы, но поскольку в момент обучения студенты программировать не умеют, их можно учить только на маленьких программах.

Подразумевается, что большие программы разрабатываются в течение длительного времени. С ними работает большой коллектив программистов. Эти программы долго эксплуатируются и переделываются. Маленькие – учебные – программы разрабатываются одним человеком за короткий срок и сразу выбрасываются.

Многие требования, естественные для больших программ, для маленьких программ кажутся (а иногда и действительно являются) ненужными. Но, несмотря на это, указанные требования приходится соблюдать.

Особенность разработки учебных программ состоит в том, что результатом студенческого труда являются вовсе не написанные программы, а те знания, умения и навыки, полученные студентами в процессе разработки этих учебных программ.

Противоречие, которое возникает при работе над любой программой (как учебной, так и производственной), заключается в удобстве двух противоположных действий: записи и чтения. При написании программы желательно, чтобы запись была более короткой. А при чтении ее – чтобы текст был более расшифрованным. Поскольку пишут программу один раз, а читают – много, это противоречие имеет смысл разрешить в пользу чтения. Лучше один раз потратить усилия на написание, чем много раз тратить их на чтение неразборчивого текста.

Заметим, что для большой программы выбор в пользу удобства чтения представляется вполне естественным. Для маленькой учебной программы такой выбор, вообще говоря, "навязывается извне".

4. Правила оформления программ

Хорошим стилем программирования считается соблюдение следующего набора правил.

Общие требования

1. Программа должна быть оформлена в одном стиле: если в каких-то правилах допускается вариативность, то для конкретной программы должен быть выбран какой-то один вариант, и именно он должен применяться во всех случаях.
2. Строки программы должны быть видны целиком (не должны уходить за правый край экрана). Более точно, длина строки не должна превышать N символов, где значение N настраивается.
3. Для выделения вложенных конструкций следует использовать абзацные отступы. Это относится как к операторам, так и к описаниям.
4. Абзацный отступ может варьироваться от 2 до 5 позиций. Отступ меньше 2 позиций не обеспечивает удобства чтения. Отступ больше 5 позиций приводит к тому, что текст программы быстро смещается вправо и уходит за край экрана.
5. Размер отступа настраивается, но все абзацные отступы в программе должны быть одинаковыми.

Идентификаторы

6. Имя должно содержать не менее 3 символов: tek, max, sum, LenLine.
7. Имя не должно совпадать по написанию или звучанию с другими именами (Ol и OI, max и maks, fon и phone) и служебными словами (than, functions).
8. Следует избегать комбинации трудноразличимых символов o/0 (буква "o" и нуль), 1/I/l (единица, латинские буквы "и" и "л"): a01 – ao1, I1 – l1.

Комментарии

9. Строки с комментариями должны составлять не менее трети строк программы.

Операторы

10. В одной строке записывается только один оператор.
11. Соответствующие друг другу операторные скобки begin и end должны располагаться на одном экране. Более точно, требуется, чтобы между begin и end было не более N строк, где N – настраиваемый параметр.

Это требование связано с тем, что восприятие алгоритма очень существенно зависит от того, можно ли охватить его одним взглядом или требуется листание страниц. Если сравнить время на разработку двух процедур, таких что алгоритм первой целиком помещается на экране, а алгоритм второй превышает размер экрана на несколько строк, то окажется, что время, требуемое на вторую процедуру, в разы превышает время, требуемое на первую. Необходимость листания резко усложняет восприятие и замедляет работу. Поэтому надо стремиться к тому, чтобы операторная часть процедуры целиком размещалась на экране. Эта идея может потребовать некоторого смягчения других требований, прежде всего требования размещать в каждой строке только один оператор.

12. Служебное слово end должно быть выровнено по соответствующей объемлющей конструкции.

В операторе ветвления слово end выравнивать по слову case. Варианты записывать с отступом с одного и того же уровня:

```
case color of
   red: S1;
   green, blue: S2;
   else S3
end
```

Для условных операторов операторные скобки располагать так:

```
if C1 then begin
   s1;
   s2
   end
else begin
   s3;
   s4
end {if C1}
```

Для цикла с предусловием операторные скобки располагать так:

```
while a > b do begin
   s1;
   s2
end; {while a > b }
```

Для цикла со счетчиком операторные скобки располагать так:

```
for k:= m to n do begin
   S1;
   S2
end; {for k}
```

Такое расположение end'а связано с неудачным синтаксисом языка Паскаль. В Паскале существуют позиции, в которых может стоять только один оператор. Именно поэтому необходим составной оператор. Эти конструкции и потребность в составном операторе Паскаль унаследовал в 1968 г. от Алгола-60. В современных языках составной оператор стал не нужен, поскольку синтаксис позволяет в любом месте, где может стоять один оператор, поставить их несколько. Заметим, что по этому пути пошел и автор Паскаля Н.Вирт в своем следующем языке – Модула-2.

Формально end закрывает составной оператор, начинающийся begin'ом. Но фактически открывающей скобкой является не begin, а начальная лексема объемлющего оператора: if, while, for. Именно по ней и предлагается выравнивать end.

Дозволяется размещать begin не в строке с заголовком объемлющего оператора, а на следующей строке:

```
if C1 then
   begin
   s1;
   s2
   end
else
   begin
   s3;
   s4
end {if C1}
while a > b do
   begin
   s1;
   s2;
end; {while a > b }
for k:= m to n do
   begin
   S1;
   S2
end; {for k}
```

Однако, учитывая требование размещать begin и end на одном экране, трата

на begin целой строки представляется слишком расточительной.

13. Не должно быть лишних (идущих подряд) символов "точка с запятой".
14. Не должно быть лишних переносов внутри конструкций (например, внутри заголовков процедур или циклов for).

Вообще говоря, список требований и рекомендаций, выдаваемых студентам, более обширен. Однако не все эти требования удается проверить автоматически. Например, от студентов требуется использовать каждую переменную только для одной цели, описывать ее как можно ближе к тому месту, где она нужна, и уничтожать как можно скорее после того, как она перестала быть нужной.

Соблюдение этих требований существенно упрощает отладку и модификацию программы. Но проверить их выполнение автоматически не представляется возможным.

5. Реализация StyleChecker

StyleChecker разработан в Windows XP, реализован в виде отдельного приложения, не требует специальной установки, но во время работы записывает информацию на диск (т.е. должен иметь право записи файлов).

StyleChecker обладает интуитивно ясным интерфейсом, выполненным в традиционном "майкрософтовском" духе.

Пример работы StyleChecker'а приведен на рис. 3.

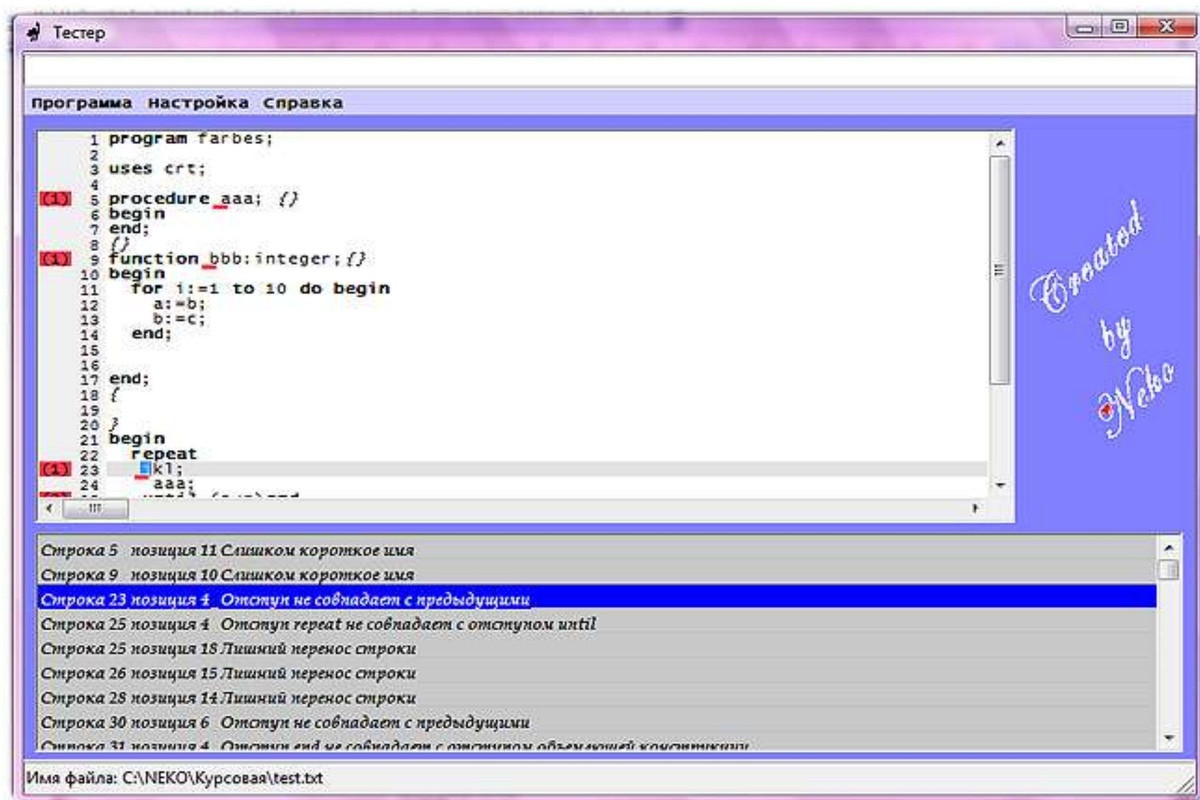

Рис. 3. *Пример работы StyleChecker'а*

После запуска StyleChecker'а пользователь может загрузить в него текстовый файл с программой и выполнить проверку. Места стилевых ошибок отмечаются в тексте программы. Рядом с номером строки выдается количество ошибок в этой строке. Диагностика выдается в отдельном окне. Продвижение по окнам с текстом программы и с диагностикой синхронизированы в обе стороны: при движении по тексту выделяется сообщение об ошибке, относящееся к текущей позиции текста; при движении по сообщениям текст позиционируется на место, соответствующее текущему сообщению.

В случае многократного повторения одной и той же ошибки после N сообщений диагностика укорачивается (параметр N настраивается). Считается, что, встретив в первый раз сообщение об ошибке, студент разберется в ее причине. И в дальнейшем ему уже не понадобятся подробные разъяснения, а будет достаточно простого напоминания.

В качестве ограничений реализации следует указать, что StyleChecker рассчитан на начинающих программистов, которые не используют наиболее "продвинутые" возможности современных расширений Паскаля. Поэтому он не работает с классами, модулями, событиями и пр. Создатели StyleChecker исходили из тех соображений, что проверять правильность оформления имеет смысл только для синтаксически правильных программ. (Исправление синтаксических ошибок потребует изменения текста программы). Поэтому, встретив в программе синтаксическую ошибку, StyleChecker прекращает проверку и ограничивается тем, что отмечает ошибочную часть программы.

В настоящее время StyleChecker внедряется в учебный процесс Пермского госуниверситета и Пермского филиала ВШЭ. Система предлагается всем желающим для бета-тестирования.

Automation of checking style of the beginner's programs

D. A. Vasenina, M. A. Plaksin
Perm State University, Russia, 614990, Perm, Bukireva st., 15
natsume_maia@mail.ru; (342) 2-396-594
State University Economy School, Russia, 614070, Perm, Student’s st., 38
mapl@list.ru; (342) 2-396-594

The program for automation of the checking good style of the beginner's programs written in language Pascal is described. The program allows to automate a part of work of the teacher on training to bases of programming.

Key words: *computer programming; style of computer programming; automation learning.*