# Лабораторная работа № 1.1. Раскрутка самоприменимого компилятора

Коновалов А.В., Скоробогатов С.Ю.

29 июня 2016

## 1 Цель работы

Целью данной работы является ознакомление с раскруткой самоприменимых компиляторов на примере модельного компилятора.

## 2 Исходные данные

### Вариант P5

В качестве модельного выберем компилятор P5 языка Pascal, разработанный С. Муром[1]. Входным языком компилятора является язык Pascal, соответствующий стандарту ISO 7185, а целевым языком — псевдокод, который может быть исполнен специальным интерпретатором.

Исходный текст компилятора P5 составлен на языке Pascal и удовлетворяет стандарту ISO 7185. Тем самым, компилятор является самоприменимым.

Исходные данные для выполнения лабораторной работы в операционной системе Linux представлены следующим набором файлов:

- `pcom.pas` — исходный текст компилятора P5;
- `pcom` — исполнимая версия компилятора P5, полученная путём компиляции исходного текста компилятора с помощью `gpc` (GNU Pascal Compiler);
- `pint` — интерпретатор псевдокода, предназначенный для выполнения программ;
- `iso7185.pdf` — текст стандарта ISO 7185:1990[2];
- `hello.pas` — программа, предназначенная для проверки работоспособности компилятора.

[1] Scott A. Moore. The P5 compiler. – URL: http://www.moorecad.com/standardpascal/p5.html.

[2] ISO 7185:1990: Information technology – Programming languages – Pascal. – Geneva, Switzerland: International Organization for Standardization, 1990.

### Вариант BeRo/BeRo64

В качестве модельного выберем компилятор BeRo Tiny Pascal, разработанный Бенжамином Рузо (Benjamin Rosseaux). Входным языком компилятора является язык Pascal, совместимый с диалектами Delphi 7 и FreePascal $\geq$ 3.0, а целевым языком — исполнимый код Win32.

На кафедре ИУ9 данный компилятор был портирован на платформы Linux x64 и macOS.

Исходный текст компилятора составлен на языке Pascal, совместимом с подмножеством диалектов Delphi 7 и FreePascal $\geq$ 3.0, при этом сам реализован на этом подмножестве. Тем самым, компилятор является самоприменимым.

Исходные данные для выполнения лабораторной работы в операционной системе Windows представлены следующим набором файлов:

- `btpc.pas` — исходный текст компилятора BeRo Tiny Pascal;
- `btpc.exe` — исполнимая версия компилятора, полученная путём раскрутки;
- `hello.pas` — программа, предназначенная для проверки работоспособности компилятора.

Исходные данные для выполнения лабораторной работы в операционных системах Linux и macOS представлены следующим набором файлов:

- `btpc64.pas`/`btpc64macOS.pas` — исходный текст портированного компилятора BeRo Tiny Pascal;
- `btpc64`/`btpc64macOS` — исполнимая версия компилятора, полученная путём раскрутки;
- `hello.pas` — программа, предназначенная для проверки работоспособности компилятора.

## 3 Использование компилятора

### Использование `pcom` и `pint`

Исполнимая версия компилятора P5 берёт исходный текст компилируемой программы из стандартного потока ввода и записывает порождаемый псевдокод в файл с именем `prr`. Тем самым, для компиляции программы `hello.pas` нужно выполнить команду

```
./pcom <hello.pas
```

Интерпретатор `pint` считывает псевдокод из файла с именем `prd`, поэтому перед его запуском требуется переименовать файл `prr` в `prd`:

```
mv prr prd
```

```
./pint
```

Исполнимую версию компилятора P5 можно применить к его исходному тексту:

```
./pcom <pcom.pas
```

После этого для компиляции программы `hello.pas` можно использовать компилятор P5, представленный в псевдокоде. Для этого нужно запустить компилятор с помощью `pint`:

```
mv prr prd

./pint <hello.pas
```

Более того, можно ещё раз откомпилировать `pcom.pas` с помощью компилятора P5, представленного в псевдокоде. Для этого нам потребуется выполнить команду

```
./pint <pcom.pas
```

Отметим, что последняя команда работает достаточно длительное время[3]. После её выполнения можно убедиться, что файлы `prd` и `prr` совпадают с точностью до пробельных символов. Сравнить их на идентичность можно при помощи комадны

```
diff -uw prd prr
```

### Использование Btpc на Windows

Компилятор берёт исходный текст со стандартного ввода и в случае успешной компиляции записывает порождённый двоичный код в стандартный вывод. Тем самым, для компиляции программы `hello.pas` нужно выполнить команду:

```
btpc <hello.pas >hello.exe
```

При наличии синтаксической ошибки в коде компилятор записывает в стандартный вывод вместо двоичного кода сообщение об ошибке. Признаком того, что компиляция прошла неудачно является малый размер целевого файла (в данном примере `hello.exe`) — менее 100 байт. Для того, чтобы посмотреть размер файла, можно выполнить команду `dir`. Для просмотра сообщения об ошибке нужно выполнить команду:

```
type hello.exe
```

Для выполнения одного шага раскрутки используется команда

```
btpc <btpc.pas >btpc_new.exe
```

После её выполнения можно убедиться, что файлы `btpc.exe` и `btpc_new.exe` идентичны при помощи команды

```
fc /b btpc.exe btpc_new.exe
```

[3] На кафедре ИУ9 разработана более быстрая реализация интерпретатора псевдокода P5: https://github.com/bmstu-iu9/P5-Interpreter

### Использование Btpc64 на Linux и macOS

Компилятор берёт исходный текст со стандартного ввода и в случае успешной компиляции записывает порождённый двоичный код в стандартный вывод. Тем самым, для компиляции программы `hello.pas` нужно выполнить команду:

```
./btpc64 <hello.pas >hello
```

При наличии синтаксической ошибки в коде компилятор записывает в стандартный вывод вместо двоичного кода сообщение об ошибке. Признаком того, что компиляция прошла неудачно является малый размер целевого файла (в данном примере `hello`) — менее 100 байт. Для того, чтобы посмотреть размер файла, можно выполнить команду `ls -l`. Для просмотра сообщения об ошибке нужно выполнить команду:

```
cat hello
```

Для выполнения одного шага раскрутки используется команда

```
./btpc64 <btpc64.pas >btpc64_new
```

После её выполнения можно убедиться, что файлы `btpc64` и `btpc64_new` идентичны при помощи команды

```
cmp btpc.exe btpc_new.exe
```

## 4 Задание

Выполнение лабораторной работы заключается в осуществлении одного шага раскрутки самоприменимого компилятора и состоит из нескольких этапов:

### Вариант P5

1. добавление во входной язык компилятора P5 новых возможностей (см. Варианты заданий) путём редактирования его исходного текста, в результате чего должен получиться файл `pcom2.pas` — следует **сначала** скопировать `pcom.pas` в `pcom2.pas`, **а потом** вносить в него правки;

2. компиляция `pcom2.pas`, которая может осуществляться как бинарной версией компилятора, так и версией, представленной в псевдокоде (бинарная — быстрее);

3. проверка работоспособности `pcom2.pas` на небольшой программе, в которой обязательно должны использоваться новые возможности языка;

4. внесение изменений в `pcom2.pas`, связанных с использованием новых возможностей языка, и сохранение новой версии исходного текста компилятора в файле `pcom3.pas` (**сначала** нужно скопировать `pcom2.pas` в `pcom3.pas`, **затем** вносить изменения);

5. завершение шага раскрутки путём компиляции `pcom3.pas` с помощью полученного на этапе 2 псевдокода компилятора;

6. разница между файлами `pcom.pas` и `pcom2.pas` должна демонстрировать изменения, внесённые в логику работы компилятора;

7. разница между файлами `pcom2.pas` и `pcom3.pas` должна демонстрировать новые возможности языка.

**Вариант с Btpc/Btpc64**

1. добавление во входной язык компилятора `btpc` новых возможностей (см. Варианты заданий) путём редактирования его исходного текста, в результате чего должен получиться файл `btpc2.pas` — следует **сначала** скопировать `btpc.pas` в `btpc2.pas` (на Linux/macOS: `btpc64.pas` в `btpc64-2.pas`), **а потом** вносить в него правки;

2. компиляция `btpc2.pas` (`btpc64-2.pas`), в результате которой должен получиться файл `btpc2.exe` (`btpc64-2`);

3. проверка работоспособности `btpc2.exe` на небольшой программе, в которой обязательно должны использоваться новые возможности языка;

4. внесение изменений в `btpc2.pas` (`btpc64-2.pas`), связанных с использованием новых возможностей языка, и сохранение новой версии исходного текста компилятора в файле `btpc3.pas` (`btpc64-3.pas`) (**сначала** нужно скопировать `btpc2.pas` в `btpc3.pas`, **затем** вносить изменения);

5. завершение шага раскрутки путём компиляции `btpc3.pas` (`btpc64-3.pas`) с помощью полученного на этапе 2 файла `btpc2.exe` (`btpc64-3`);

6. разница между файлами `btpc.pas` и `btpc2.pas` должна демонстрировать изменения, внесённые в логику работы компилятора;

7. разница между файлами `btpc2.pas` и `btpc3.pas` должна демонстрировать новые возможности языка.

**Просмотр различий между файлами**

На операционной системе Windows для просмотра различий между файлами рекомендуется использовать встроенную утилиту `fc`:

```
fc pcom.pas pcom2.pas
fc btpc.pas btpc2.pas
```

На операционных системах Linux и macOS для просмотра различий между файлами рекомендуется использовать встроенную утилиту `diff` с ключом `-u`:

```
diff -u pcom.pas pcom2.pash
diff -u btpc64.pas btpc64-2.pas
```

## 5 Варианты заданий

Номер варианта определяется по формуле

$$\text{номер варианта} = \text{номер зачётки} \% 40 + 1$$

Где номер зачётки — число после буквы факультета. Например, номер зачётки — 05M136, номер варианта — $136 \% 40 + 1 = 16 + 1 = 17$.

1. Компилятор **P5**. Заменить операторы `div` и `mod` на `//` и `%` соответственно.
2. Компилятор **BeRo**. Заменить операторы `div` и `mod` на `//` и `%` соответственно (однострочные комментарии перестанут поддерживаться).
3. Компилятор **P5**. Не разрешать комментариям, начинающимся с `(*`, заканчиваться на `}`, а комметариям, начинающимся с `{`, заканчиваться на `*)`.
4. Компилятор **BeRo**. Разрешать комментариям, начинающимся с `(*`, заканчиваться на `}`, а комметариям, начинающимся с `{`, заканчиваться на `*)`.
5. Компилятор **P5**. Сделать так, чтобы можно было использовать идентификаторы любой длины, но при этом символы идентификатора, начиная с одиннадцатого, не учитывались.
6. Компилятор **BeRo**. Выводить сообщение об ошибке при превышении длины идентификатора 35 символов.
7. Компилятор **P5**. Добавить в язык шестнадцатеричные константы вида `0×12ABcd`.
8. Компилятор **BeRo**. Заменить шестнадцатеричные константы вида `$12ABcd` на константы вида `0×12ABcd`.
9. Компилятор **P5**. Добавить в строковые литералы Escape-последовательности `\a`, `\b`, `\t`, `\\`.
10. Компилятор **BeRo**. Добавить в строковые литералы Escape-последовательности `\a`, `\b`, `\t`, `\\`.
11. Компилятор **P5**. Сделать так, чтобы символы **..** и **:** были взаимозаменяемыми.
12. Компилятор **BeRo**. Сделать так, чтобы символы **..** и **:** перестали быть взаимозаменяемыми.
13. Компилятор **P5**. Обеспечить возможность использования в числовых литералах незначимый знак `_` (например, число 10 000 можно записать и как `10000`, и как `10_000`, и как `100__00`).
14. Компилятор **BeRo**. Обеспечить возможность использования в числовых литералах незначимый знак `_` (например, число 10 000 можно записать и как `10000`, и как `10_000`, и как `100__00`).
15. Компилятор **P5**. Сделать так, чтобы символы в строке программы, расположенные справа от 80-й позиции, не учитывались (считались комментарием).
16. Компилятор **BeRo**. Сделать так, чтобы символы в строке программы, расположенные справа от 110-й позиции, не учитывались (считались комментарием).

17. Компилятор **P5**. Разрешить использовать знак `..` вместо ключевого слова `to` при записи цикла `for`. При этом использование слова `to` не запрещается.
18. Компилятор **BeRo**. Разрешить использовать знак `..` вместо ключевого слова `to` при записи цикла `for`. При этом использование слова `to` не запрещается.
19. Компилятор **P5**. Сделать идентификаторы и ключевые слова чувствительными к регистру.
20. Компилятор **BeRo**. Сделать идентификаторы и ключевые слова чувствительными к регистру.
21. Компилятор **P5**. Добавить однострочный комментарий, начинающийся с символа `?`. Т. е. суффикс строки программы, расположенный после символа `?`, должен считаться комментарием.
22. Компилятор **BeRo**. Добавить однострочный комментарий, начинающийся с символа `?`. Т. е. суффикс строки программы, расположенный после символа `?`, должен считаться комментарием.
23. Компилятор **P5**. Добавить синонимы `~`, `&` и `|` для операторов `not`, `and` и `or` соответственно. При этом операторы `not`, `and` и `or` остаются допустимыми.
24. Компилятор **BeRo**. Добавить синонимы `~`, `&` и `|` для операторов `not`, `and` и `or` соответственно. При этом операторы `not`, `and` и `or` остаются допустимыми.
25. Компилятор **P5**. Сделать так, чтобы целочисленные константы, выходящие за границы допустимого интервала, считались равными нулю.
26. Компилятор **BeRo**. Сделать так, чтобы целочисленные константы, выходящие за границы допустимого интервала, считались равными нулю.
27. Компилятор **P5**. Обеспечить возможность использования символа `@` в идентификаторах.
28. Компилятор **BeRo**. Обеспечить возможность использования символа `@` в идентификаторах.
29. Компилятор **P5**. Добавить в язык двоичные константы вида `0b10010`.
30. Компилятор **BeRo**. Добавить в язык двоичные константы вида `0b10010`.
31. Компилятор **P5**. Заменить запись операции $\diamond$ на $\neq$.
32. Компилятор **BeRo**. Заменить запись операции $\diamond$ на $\neq$.
33. Компилятор **P5**. Заменить ключевые слова `begin` и `end` на `{` и `}` соответственно. При этом ключевые слова `begin` и `end` остаются допустимыми.
34. Компилятор **BeRo**. Заменить ключевые слова `begin` и `end` на `{` и `}` соответственно. При этом ключевые слова `begin` и `end` остаются допустимыми.
35. Компилятор **P5**. Разрешить возможность не писать ключевое слово `then` после условия в блоке `if`.
36. Компилятор **BeRo**. Разрешить возможность не писать ключевое слово `then` после условия в блоке `if`.
37. Компилятор **P5**. Разрешить использовать ключевое слово to вместо знака `..` при записи типа диапазона. При этом использование знака `..` не запрещается.
38. Компилятор **BeRo**. Разрешить использовать ключевое слово to вместо знака `..` при записи типа диапазона. При этом использование знака `..` не запрещается.
39. Компилятор **P5**. Заменить запись оператора присваивания на `::=`.
40. Компилятор **BeRo**. Заменить запись оператора присваивания на `::=`.