The Pascal Programming Language

a report by B.A.C. Schopman
for the course
Principles of Programming Languages
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
<td>3</td>
</tr>
<tr>
<td>History, purpose and use</td>
<td>3</td>
</tr>
<tr>
<td><strong>Pascal vs. C</strong></td>
<td>4</td>
</tr>
<tr>
<td>Differences</td>
<td>4</td>
</tr>
<tr>
<td>Parameters</td>
<td>4</td>
</tr>
<tr>
<td>Statement syntax</td>
<td>4</td>
</tr>
<tr>
<td>Operators</td>
<td>4</td>
</tr>
<tr>
<td>Comments</td>
<td>5</td>
</tr>
<tr>
<td>Case sensitivity</td>
<td>5</td>
</tr>
<tr>
<td>If-statement</td>
<td>5</td>
</tr>
<tr>
<td>Strings</td>
<td>5</td>
</tr>
<tr>
<td>Similarities</td>
<td>5</td>
</tr>
<tr>
<td>Blocks</td>
<td>5</td>
</tr>
<tr>
<td>Variable definitions</td>
<td>5</td>
</tr>
<tr>
<td>In-line assembly</td>
<td>5</td>
</tr>
<tr>
<td><strong>Trade-offs</strong></td>
<td>6</td>
</tr>
<tr>
<td>Ambiguity vs. cluttered grammar</td>
<td>6</td>
</tr>
<tr>
<td>Case #1: Precedence</td>
<td>6</td>
</tr>
<tr>
<td>Case #2: Associativity</td>
<td>6</td>
</tr>
<tr>
<td>Case #3: Dangling else</td>
<td>6</td>
</tr>
<tr>
<td>Conclusion</td>
<td>7</td>
</tr>
<tr>
<td>Early vs. late binding</td>
<td>8</td>
</tr>
<tr>
<td>Strong vs. weak typing</td>
<td>8</td>
</tr>
<tr>
<td>Static vs. dynamic typing</td>
<td>8</td>
</tr>
<tr>
<td>Garbage collection or not</td>
<td>8</td>
</tr>
<tr>
<td>Single vs. multiple inheritance</td>
<td>8</td>
</tr>
<tr>
<td><strong>Polymorphism</strong></td>
<td>9</td>
</tr>
<tr>
<td>Procedure/function name overloading</td>
<td>9</td>
</tr>
<tr>
<td>Operator overloading</td>
<td>9</td>
</tr>
<tr>
<td>Coercion</td>
<td>9</td>
</tr>
<tr>
<td><strong>Appendix A: a sample program</strong></td>
<td>11</td>
</tr>
<tr>
<td>Circle</td>
<td>11</td>
</tr>
<tr>
<td>Overloading</td>
<td>12</td>
</tr>
<tr>
<td><strong>Appendix B: information sources</strong></td>
<td>13</td>
</tr>
</tbody>
</table>
Introduction
This report discusses the Pascal programming language and how it differentiates from similar programming languages. The most obvious programming language to compare Pascal to is C, so I will compare Pascal to C in the first chapter.

There is a lot to say about a programming language. Unfortunately it is too much to discuss in a short paper as this one. Due to time shortage this paper may seem a bit hectic, which is because I have tried to explain most significant issues compactly.

In the second chapter I will discuss significant tradeoffs to the Pascal Programming language. Finally, in the last chapter polymorphism will be briefly discussed.

Many dialects of the Pascal programming language were developed since it was created. Not all features will be discussed, such as object orientation, because this paper will be devoted to the essential trade-offs. The version of Pascal I will use to test features is Free Pascal, which can be found on http://www.freepascal.org.

History, purpose and use
In 1971 Professor Nicklaus Wirth created a programming language based on Algol. He named this programming language Pascal, in honor of the mathematician Blaise Pascal, pioneer of the mechanical calculator. Professor Wirth made this programming language for students to learn structured programming. That is why Pascal is a generally clearly structured programming language, which relatively easily to read (compared to C code, which can be totally unreadable). Because Pascal meant to be a structured programming language, the code is relatively easily efficiently translated to assembly (or machine code). That is why it was used for system programming purposes by Macintosh developers in the early versions of Mac OS. Therefore Pascal is still very much alive as a programming language on the Macintosh platform (just take a look at the frontpage of http://www.pascal-central.com/ for an illustration).
Pascal vs. C

I have not as much programming experience with Pascal as with C, but I find that Pascal is very much like C. Although there are some differences in syntax, there are a lot of similarities in semantics.

Differences

Parameters

The first difference, which is very important, is that in Pascal parameters of procedures and functions can be passed by value-result. This means a number of variables can be easily changed by a function instead of that a function only returns one unit as in C. So, for instance, a function that calculates the diameter, area and circumference of a circle can be defined as follows:

```pascal
procedure Circle(radius: integer; {input}
    var diameter: integer; var area, circumference: real);

begin
    diameter := radius * 2;
    area := 3.14*2*radius;
    circumference := 3.14 * 3.14 * radius
end;
```

This function takes an integer which is called radius and calculates the diameter, area and circumference. The program in which this function is used is included in Appendix A as an example of a working Pascal program.

Statement syntax

A strange difference of Pascal to all other programming languages I know is that the last statements of a function or program do not have to be closed with a semicolon. The semicolon can be added if the programmer wishes to be consequent. This does not add to the structure of the programming language and can be seen as a shortcoming, because if the student learning structures programming adds some lines at the end of the program and does not add a semicolon the compiler will complain. The sample program of Appendix A shows that the last lines do not need semicolons.

Operators

There are quite a lot of differences in operators, which are shown in the table below. The assignment operator is more natural in Pascal then in C, but as it is used more often then the equivalence operator the choice in C is better, as it is a single character instead of two. The “not equivalence operator” of Pascal has a different interpretation, as it is supposed to be read as “is smaller or greater” instead of “is not”.

The final two operators are both important when using pointers, which is not generally used. Pascal is moreover used as a language to build clearly structured programs and not systems programming.

<table>
<thead>
<tr>
<th>Function</th>
<th>in Pascal</th>
<th>in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment operator</td>
<td>:=</td>
<td>=</td>
</tr>
<tr>
<td>Equivalence operator</td>
<td>=</td>
<td>==</td>
</tr>
<tr>
<td>Not equivalence operator</td>
<td>&lt;&gt;</td>
<td>!=</td>
</tr>
<tr>
<td>Address operator</td>
<td>@</td>
<td>&amp;</td>
</tr>
<tr>
<td>Pointer operator</td>
<td>^</td>
<td>*</td>
</tr>
</tbody>
</table>
Comments
Another important difference is the comment operators. In Pascal text can be made a comment by entering it between curly brackets or parentheses-stars.

```
{ These are comments }
(* just as these *)
```

Because C uses curly brackets as block operators, this is an important difference. Where in C the beginning and ending of a block are respectively indicated by { and }, in Pascal it is indicated by the keywords begin and end, as such:

```
Begin
  (* instead of a { in C *)
  This is a block:
  Any number of statements can be inserted here:
  (* instead of a } in C *)
End
```

Case sensitivity
Another syntactic difference is that Pascal is not case sensitive. This lowers the learning curve a bit, which makes it more suitable for educational purposes then C.

If-statement
The if-statement of Pascal looks a little bit different then the one in C:

<table>
<thead>
<tr>
<th>If-statement in Pascal</th>
<th>if-statement in C</th>
</tr>
</thead>
<tbody>
<tr>
<td>If &lt;boolean expression&gt; then &lt;expression/block&gt; else &lt;expression/block&gt;</td>
<td>if(&lt;boolean expression&gt;) &lt;expression/block&gt; else &lt;expression/block&gt;</td>
</tr>
</tbody>
</table>

The if-statement in Pascal looks a little more “natural” (i.e. easier to understand for someone who does not have programming experience). This also lowers the learning curve slightly.

Strings, …
There are too many differences to name them all, but a last difference I would like to mention is that strings are defined not between double quotes ("), but single quotes (’).

Similarities

Blocks
There are quite some similarities with C and most of them are in semantics. For instance, blocks work the same, with the only difference that in Pascal a block must be made inside an if-statement, for-statement or such. In C a block can be placed anywhere for any reason.

Variable definitions
Another equivalence of Pascal and C is that variables have to be defined at the beginning of the program. This is an important similarity, because it means the required memory space is allocated at the beginning of executing the program.

In-line assembly
Another important similarity is that in both Pascal and C is it possible to insert assembly code in your program. This makes it possible to execute operations that would otherwise be impossible, but can make it very hard to read a problem. Another downside is that it makes a program with in-line assembly totally platform dependant.
Trade-offs
There is no such thing as the best programming language. There are several kinds of programming language, such as imperative, functional and logic languages. Pascal is an imperative language, because the programmer exactly defines what the computer must do when the program is executed.
There are certain trade-offs that are significant for virtually every programming language. When a language is designed decisions must be made, which have impact on how efficient the language is and/or how easy the language is to use or read.

Ambiguity vs. cluttered grammar

Case #1: Precedence
There are only four precedence levels in Pascal:

Highest level: Not @
Second level: * / div mod and shl shr as
Third level: + - or xor
Lowest level: < <> <= >= in is

Because the or-operator is on a higher level than relational operators, a Boolean expression such as i<3 or i<9 will be found erroneous by the compiler. With parentheses the programmer must indicate that the desired operation is a disjunction of two equations (i<3) or (i<9).

Case #2: Associativity
Pascal has left operator associativity. I tested this using the following program:

```pascal
program associativity(input, output);
begin
  writeln('25 + 13 mod 7 = ', 25 + 13 mod 7);
  writeln('25 * 13 mod 7 = ', 25 * 13 mod 7);
end.
```

The output of this program is:

```
12:13:26 $ ./associativity
25 + 13 mod 7 = 31
25 * 13 mod 7 = 3
```

The output is exactly as expected, because the first expression is equivalent to 25 + (13 mod 7), as mod has a higher precedence then +. But in the second example all operators have an equal precedence level, so because of the left associativity that statement is equivalent to (25 * 13) mod 7.

Case #3: Dangling else
There is a “dangling else”-problem in Pascal, which works exactly the same as in C. Here is the program in which I tested the if-then-else statement:

```pascal
program overloading(input, output);
```
var
  i: integer;
begin
  i := 123;
  { 1: if-then-else statement with dangling else }
  if i < 0
    then
      if i > 100
        then writeln('a')
        else writeln('dangling else')
      { another else-statement could be added here };
  { 2: if-then-else using block to prevent ambiguity }
  if i < 0
    then
      begin
        if i < 1000
          then writeln('case 1')
      end
      else writeln('case 2');
end.

The output is of this program is as follows, but try to think about what the output should be before you read this.
12:51:30 $ ./precedence
   case 2

In the 1st if-then-else statement there is a dangling else. The else could either belong to the first or second if-condition. In case it would belong to the first if, then ‘dangling else’ would be printed, since i>0. But that is not true, so the whole body of the if-statement will not be executed.

In the 2nd if-then-else statement a block is used to prevent ambiguity. The begin and end keywords in Pascal are the same as the { and } in C: they define the beginning and end of a block. So here the else-clause is not dangling anymore, because it can only belong to the first if-condition.

Conclusion
The small amount of precedence levels makes the language less complex, so easier to understand and faster. The downside is that more parentheses have to be used to express Boolean expressions, such as (i<3) or (i<9).
The associativity is just as one would expect, because left associativity is also the standard in mathematics.
The dangling else problem is solved with blocks, just like in C. It is up to the programmer to make it readable, with correct indenting and blocks. It makes the implementation of the programming language very efficient.
The conclusion is that Pascal has a slightly cluttered grammar, providing the tools to eliminate ambiguity, but simply structured code does not have to look confusing.
Early vs. late binding

Pascal generally uses early binding. That means that types of variables, constants, methods, etc are known at compile time. In most cases the compiler will show an error and stop compiling when an incompatible binding is made.

Strong vs. weak typing

Pascal is a strongly typed language. That means that for instance, that when \( i \) is an integer and a program contains the following statement:

\[
    i := 'this is a string';
\]

The compiler will show the following error and not continue compiling.

```
types_integer.pp(9,7) Error: Incompatible types: got "ShortString"
expected "SmallInt"
```

But the compiler will not show an error in every case of invalid assignments. For instance, when we would assign a very large number to the same integer \( i \), the compiler will show a warning. A warning does not cause the compiler to stop and the program will be constructed. But if for instance the following statements are included in a program:

```
    i := 3248756;
    writeln(i);
```

The compiler will show the following warning:

```
 types_integer.pp(7,7) Warning: range check error while evaluating constant
```

When the 2\(^{nd}\) line will be executed, the number \(-28044\) will be shown, so data loss will occur.

Strong typing makes the Pascal programming language less flexible, but makes much more reliable. It forces the programmer to be conscious of what the boundaries of data types are and gives the chance to fix problems before program execution.

Static vs. dynamic typing

Pascal uses static typing, because in dynamic typing type-checking happens at runtime. That would mean a line like \( i := 4 + 'string'; \) would be possible, which is not true.

Garbage collection or not

Pascal does not have a garbage collector. This is because Pascal is made to be a simple and efficient programming language. Object orientation was not an issue at language design time and dynamically allocated variables are generally not used, as it is not made for advanced programming. Therefore there is no need for a garbage collector.

Single vs. multiple inheritance

Since Pascal is not a language in which object orientation is a standard, this trade off is not significant. However, in a object-oriented extension to Pascal multiple inheritance was chosen. The document on this extension can be found here:

http://www.pascal-central.com/OOE-stds.html#sect-6.3.2
Polymorphism

Procedure/function name overloading
In the version of Pascal I have been using procedure overloading is possible. The program “procedure overloading” in Appendix A illustrates this.

Operator overloading
Predefined operator overloading is a feature of Pascal, because the arithmetic and relational operators can be used for several data types (integer, real, etc). User-operator overloading is also possible, as it is described in the following document:
http://www.freepascal.org/docs-html/ref/refch11.html#x134-14100011

Coercion
Coercion is also tested in the overloading-program (see Appendix A). An integer can be compared with a real, which means that the integer is converted to a real (not visa versa, because real to integer translation would result in data loss).
Appendix A: a sample program

**Circle**

Program:

```
program circle(input, output);

var
  r: integer;
  d: integer;
  a: real;
  c: real;

procedure Circle(radius: integer; {input}
  var diameter: integer; var area, circumference: real);
  begin
    diameter := radius * 2;
    area := 3.14*2*radius;
    circumference := 3.14 * 3.14 * radius
  end;

begin
  r := 4;
  circle(r, d, a, c);
  writeln('radius = ',r);
  writeln('diameter = ',d);
  writeln('area = ',a:4:2);
  writeln('circumference = ', c:4:2)
end.
```

Output:

```
$ ./circle
radius = 4
diameter = 8
area = 25.12
circumference = 39.44
```
Overloading

Program
program overloading(input, output);

var
  i: integer;
  r: real;
  s: real;
  a: real;

procedure ComputeSumAve(num1, num2: Real; {input}
  var sum, average: Real); {output}
begin
  Sum := num1 + num2;
  average := sum/2
end;

procedure ComputeSumAve(num1, num2: Integer; {input}
  var sum: Integer; var average: Real); {output}
begin
  Sum := num1 + num2;
  average := sum/2
end;

begin
  r := 40.1;
  ComputeSumAve(r, 20.5, s, a);
  writeln('sum=', s:3:2, ' average=', a:3:2);
  ComputeSumAve(4.5, i, a);
  writeln('sum=', i, ' average=', a:3:2);
  writeln();
if 40 < 46.2341 then writeln('coercion is possible');
if 40 > 46.2341 then writeln('coercion is not possible');
end.

Output:
$ ./overloading
sum=60.60 average=30.30
sum=9 average=4.50
coercion is possible
Appendix B: information sources

Free Pascal
   http://www.free-pascal.org
Wikipedia
   http://en.wikipedia.org/wiki/Pascal_programming_language
Pascal central
   http://www.pascal-central.com/
Principles of Programming course

Free Pascal compiler used:
Free Pascal Compiler version 2.0.2 [2005/11/15] for powerpc
Copyright (c) 1993-2005 by Florian Klaempfl
Target OS: Darwin for PowerPC