Class 02: Detour: A Quick Introduction to Pascal

Held: Monday, 29 August 2011

Summary: We quickly explore the Pascal programming language, which will serve as our source language for the semester.

Related Pages:

- EBoard.
- Reading: Skim [The Programming Language Pascal: 1973 revised edition](#)
- Due: Introductory Survey.

Notes:

- Due tonight: Introductory Survey.
- Due Wednesday: Aho et al., Chapter 1.
- Due next Monday: Playing with Pascal. (I anticipate working on the assignment in the next day or so.)
- I’ve started to answer your questions from the introductory survey..
- Please submit your questions on homework assignments on Piazza.
- EC for Thursday’s CS Extra on Humanitarian HCI.

Overview:

- Leftovers: The Back End.
- A Short History of Pascal.
- Key Components of Imperative Languages.
- Detour: BNF.
- Overall Structure.
- Variables and Types.
- Basic Operations.
- Conditionals.
- Types.
- Looping Structures.
- Procedures.

A Short History of Pascal

- Pascal is a descendendant of Algol.
- Algol was one of the first four “big” programming languages:
  - Fortran: Designed for mathematical computation. Popular primarily in Physics.
• LISP: Designed for AI. Popular primarily at MIT.
• Algol: Designed for computer scientists to use to express algorithms. Popular primarily in Europe.
• Algol is an imperative language. You should know what that means.
• Niklaus Wirth designed Pascal with two goals in mind
  ○ Getting rid of many things he didn’t like that his colleagues had added to Algol
  ○ Creating a language good for teaching
• Many computer scientists still use a Pascal-like syntax when writing imperative algorithms.

Why Pascal?

• We’re using Pascal as our source language for a few reasons.
• It is a simple and straightforward language which should not take much time to learn.
• It was designed to be easy to compile. (We’ll see evidence of this design at various places.)
• It has fairly strong typing, and typing is useful to consider when you’re writing a compiler.
• Pascal has had enough of an influence that it’s useful for people to see it at least once in their undergraduate careers.

What Belongs in an Imperative Language?

• Pascal is an imperative language, so we should consider the key aspects of imperative languages as a way of helping us decide what to look at.
• Here are some things I expect in an imperative language:
  ○ Variables
  ○ Types for those variables (sometimes implicit, sometimes explicit)
  ○ Type definitions
• Basic operations for input, output, and assignment
  ○ A mechanism for sequencing operations
  ○ Control structures for conditional execution
  ○ Control structures for repeated evaluation
  ○ Procedures (that permit recursion)
• We’ll cover as many of these issues as we can at a high level and return to more precise details throughout the semester.

Detour: BNF

• Backus-Naur Form (BNF) is a notation for the syntax of languages.
• Basically, you give a syntactic unit and then explain how that unit is built.
• In English, we might write

```
<simple sentence> ::=<subject> <intransitive verb>
   | <subject> <transitive verb> <direct object>
   | ...
<subject> ::= <adjective>* <noun>
```
• We’ll consider BNF in depth when we do parsing.
• Note that some of the things that Wirth describes in BNF we’ll handle with a lexer, rather than a parser.

Overall Structure

• Pascal programs have an interesting structure.
• The structure looks something like the following

  program name(ports);

  const
    constant declarations

  type
    type declarations

  var
    variable declarations

  procedure and function definitions

  begin
    sequence of statements separated by semicolons
  end.

• All of the definitions and declarations are optional.

Variables and Types

• Variable (and other) names in Pascal look much like they do in C and Java: Sequences of alphanumerics that begin with an alphabetic character. (I’ll admit that I never remember whether other characters are permitted, and implementations seem to differ. The standard says only letters and numbers.)
• Variables must be declared (as in Java and C) and typed.
• Once you’ve indicated that you are declaring variables with the var keyword, you write variables names (separated by commas), a colon, and a type. For example

  var
    x, y: integer;
    z: real;

• The basic types are integer, real, and Boolean.
• The primary compound type is the array, written as

  array[lowest-index..highest-index] of type

• One can also build “records” (something like classes, but without the procedures) with
record variable-declarations end

- Pascal also makes it easy to define types that are collections of names (enumerated types). For example:

```pascal
type
  weekdays = (Monday, Tuesday, Wednesday, Thursday, Friday);
```

### Simple Statements

- Assignment is done with the `:=` operation. For example,

```pascal
x := x+1;
```

- Input is done with `read` and `readln`.

  ```pascal
  read(x);
  readln(x);
  ```

  - The extra `ln` indicates that the end-of-line character should be read.

- Output is done with `write` and `writeln`.

  ```pascal
  write(x);
  writeln(x);
  ```

  - The extra `ln` indicates that the end-of-line character should be printed.

- Neighboring statements are separated by semicolons.

- Sequences of statements can be grouped together into a compound statement by surrounding the sequence with the keywords `begin` and `end`.

### Conditionals

- Simple

  ```pascal
  if test then statement
  ```

- With alternative

  ```pascal
  if test then statement else statement
  ```

- Case statements

  ```pascal
  case exp of
    val1: statement1
    val2: statement2
    ...
    valn: statementn
  end
  ```

  - Wirth felt that the values should encompass all possible values of the expression.
  
    - Implementors have usually added a default or else case.
    - Many implementors have allowed ranges and groups of values on the left.

  - Case statements can be a lot of fun to compile. (We’ll return to the issue later in the semester.)
**Compound Types**

- The primary compound type is the array, written as
  \[
  \text{array[lowest-index..highest-index] of type}
  \]

- One can also build “records” (something like classes, but without the procedures) with
  \[
  \text{record variable-declarations end}
  \]

- Pascal also makes it easy to define types that are collections of names (enumerated types). For example
  \[
  \text{type}
  \begin{align*}
  \text{weekdays} &= \{\text{Monday, Tuesday, Wednesday, Thursday, Friday}\};
  \end{align*}
  \]

**Loops**

- “For” loops are intended only for enumerated types.
  \[
  \text{for var := starting-value to ending-value do statement}
  \]

- You can loop from high to low using \text{downto} rather than \text{to}

- In some implementations, you can specify a “step” with \text{step val}.

- Note that you can use user-defined types as well as integers.

- “While” loops have a form similar to that of C
  \[
  \text{while test do}
  \begin{align*}
  \text{statement}
  \end{align*}
  \]

- \text{Repeat} loops put the test at the end
  \[
  \text{repeat}
  \begin{align*}
  \text{series-of-statements}
  \end{align*}
  \text{until test}
  \]

**Procedures**

- Pascal differentiates procedures, which do not return values, from functions, which do.

- The form looks much like that of a program
  \[
  \text{procedure (typed-parameters)};
  \begin{align*}
  \text{var}
  \begin{align*}
  \text{variables}
  \end{align*}
  \begin{align*}
  \text{local-procedures}
  \end{align*}
  \text{begin}
  \begin{align*}
  \text{statement-list}
  \end{align*}
  \text{end}
  \]

- For functions, you must also specify a return type.
Return values from functions are treated strangely.
  - You must assign to “the function variable” (which has the same name as the function).
  - The value of that variable when the function terminates is the value of the function.