CS 4100
Pascal Highlights

March 28, 2011

Based on slides by Istvan Jonyer
Book by MacLennan

Chapter 5:
Return to Simplicity: Pascal

- 1964 IBM: PL/I (Programming Language one) evolves to be a huge language
  - Union of Fortran, Algol and COBOL (rather than their intersection)
  - Swiss Army Knife Approach
  - Language is hard to use
    - Proponents say, enough to learn subset of PL/I
    - In reality, due to feature interaction, this is not possible
- Hard (or even futile) to design to design a language that is everything to all programmers

Extensible Languages

- Another approach is to design a small 'kernel' language and make it extensible
  - Kernel provides basic functionality
  - Extensibility should please everyone

Extensions: Operators

- Operator extension (vs overload)
  - Ability to create new operators
  - Example: symmetric difference of real numbers

```
operator 2 x # y;
  value x, y; real x, y;
  begin
   return abs(x - y)
  end
```

  - Allows:

```
if l # r > 0 then …
```

- C++ has operator overload, variation of this

Extensions: Syntax

- *Syntax macros* allowed general syntax extension

```
real syntax sum from i = lb to ub of elem;
  value lb, ub;
  integer lb, ub, i; real elem;
  begin real s; s := 0;
   for i := lb step 1 until ub do
         s := s + elem;
   return s;
  end;
```

  - Allows:

```
total := sum from k = 1 to N of Wages[k];
```

5

Issues with Extensibility

- Inefficiency
  - New syntax is translated to kernel constructs
  - Inefficiencies are magnified
- Poor diagnostics
  - Compiler errors are issued at kernel-level, which may be confusing to programmer
  - Language is hard to read, since people make up their own syntax
- Upside
  - Research on minimal requirement for PL's

6

Move Toward Simplicity

- Niklaus Wirth suggests changes to Algol-60
  - Non-numeric data types
  - Removing baroque features
  - Maintain efficiency (compile and run-time)
  - Can be taught systematically
- Implements Algol-W (after changes are rejected by Algol committee)
  - Evolves into Pascal, competed in 1970

7

Pascal - 3rd Generation

- Developed 1968-1970
  - 29 page report
- Revised 1972
- International Standard 1982
- Popular teaching language

8

Pascal's Syntax

- Pascal's syntax is like Algol's (p. 171)
- Major changes
  - program … end.
  - procedure <declarations> begin <statements> end;
  - var, const, type
  - *for*-loop: simplified
  - case-statement

9

var, const, type

- const
  - Constant parameter declaration

```
const Max = 900;
```

- type
  - Type declarations introduced by "type"

```
type index = 1 .. Max;
```

- var
  - Variables declared after "var"

```
var
  i: index;
  sum, ave, val: real;
```

10

Data Structures

- Primitives are like Algol's
  - real, integer, Boolean, **char**
  - Char holds one character
    - Strings are arrays of chars

11

Enumeration Types: Issues

- Problem:
  - How to manipulate non-numeric data?
  - Mon, Tue, Wed,… Male/Female,
- Using number is very confusing (error prone)
  - today := 5; // Friday
  - tomorrow := today + 1; // next day
  - Issues: Sunday: 0 or 1? Start week with Monday?
- Assign numbers to meaningful variables
  - Mon = 1, Tue = 2, … male = 0, female = 1, …
- Security Issue: compiler allows meaningless operations
  - Year : = (month + male)/DayOfWeek

12

Enumeration Types

- Pascal introduces enumeration types

```
type
  month = (Jan, Feb, Mar, Apr, May, …);
  sex = (male, female);
var
  thisMonth : month;
  gender : sex;
begin
  thisMonth := Feb;
  gender := female;
```

- Supported operations for all enumerated types
  :=, succ, pred, =, <>, <, =, >, <=, >=

13

Enumeration Types

- Advantages
  - High level
    - Lets programmers write what they mean
  - Secure
    - Type checking is performed
    - No meaningless operations
  - Efficient
    - Allows optimization of storage
    - E.g.: Days of week can be stored in 3 bits

14

Subrange Types

- Improve security by allowing variable to take on values meaningful for their use only

```
var DayOfMonth: 1 .. 31;
type Weekday = Mon .. Fri;
```

  - Checking of valid values are checked as part of type checking
  - Many programming errors come down to subrange violations (array out of bounds)
  - Efficient: Allows compact storage of variable
  - Subranges of discrete types are allowed
    - integer, enumerated, char

15

Set Types

- Pascal provides facilities for sets

```
set of <ordinal type>
```

  - Ordinal type: enumeration, char, Boolean, subrange
  - Not integer or real

```
var S, T: set of 1..10;
```

  - S, T can hold a set of numbers between 1 and 10
    - vs a single number between 1 and 10:

```
var S, T: 1..10;
```

16

Efficiency of Sets

- Set types are restricted to be ordinal to be efficient

```
var S, T: set of 1..10;
```

  - S, T take only 10 bits to represent: 1 bit for each number
    - Bit = 0 means number is not is set
    - Bit = 1 means number is in set
  - `S := [1,2,3,5,7];`

	1	2	3	4	5	6	7	8	9	10
S =	1	1	1	0	1	0	1	0	0	0

17

Set Operations

- Initialization/Assignment

  []

```
T := [1..6];
```

- Membership

  in

```
if 4 in T then …
```

- Union, intersection, difference

  +, *, -

```
S * T, S + T, …
```

- Comparisons
  - Subset, equality, non-equality
  - <=, >=, =, <>
  - Proper subset (<) is not provided

18