WATCOM Pascal

in an

Educational Environment

Overview

- What is Pascal, and why should it be used for teaching Computer Science?
- Requirements for software used for teaching
- WATCOM Pascal

History of Pascal

- developed by Niklaus Wirth in late 1960s
- derived from ALGOL family; predecessor to Modula
- Wirth:
  "... a language suitable for teaching programming as a systematic discipline based upon certain fundamental concepts ... [and a] language which [is] both reliable and efficient"
- numerous implementations; many extensions
- language standardized by ISO and ANSI in 1983
- standards committees now working towards "standard" extensions

Features of Pascal

- small language with concise rules
- not too much to learn in order to write significant programs
- ability to describe and structure data precisely
- sequence control statements: `for`, `while`, `repeat-until`, `if`, `case`
- modular program organization: procedures (subroutines) and functions
- names may be of any length
- all items must be declared before they are used -- forces programmer to organize program
- variables must be initialized before they are used -- no "default" values

Data Organization

- variables must be declared to be of some type -- e.g.

```
var salary : real;
```

- a type states what values a variable may contain
- standard types: integer, real, char (character), Boolean
- Pascal will not permit a variable to be assigned a value of the wrong type -- prevents misuse of variables
- type compatibility
- data structuring facilities: arrays, records, pointers, programmer-defined types
- much of the power of Pascal comes from its ability to describe data precisely

Programmer-defined Types

- a type indicates what values a variable may contain
- example: integers are ...-3, -2, -1, 0, 1, 2, 3...
- Pascal allows programmer to define new types -- simply state the values that are to comprise the new type
- example:

```
type TwentiethCentury = 1900..1999;
```

- example:

```
type days = (mon, tue, wed, thu, fri, sat, sun)
```

- the type `days` consists of exactly the seven values given
- `mon`, `tue` ... are constants of the type `days`, just as 1, 2 ... are constants of integer type
- if a variable is declared to be of type `days`, then only those seven values may be assigned
- called an enumerated type

Why Use Programmer-defined Types?

- allows better description of data -- if a variable will contain only numbers from 1900 to 1999, let Pascal enforce

- if a variable is to represent days of the week, use symbolic constants -- more descriptive, much less prone to error

- for example, without enumerated types, typical approach would be to use numeric values, and agree that 1 is Monday, 2 is Tuesday etc.

- error prone: potential disaster if number 8 is used

- with enumerated types, such an error is impossible:

```
type days = (mon, tue, wed, thu, fri, sat, sun);
var payday : days;
```

  Pascal guarantees that `payday` will contain one of the seven given values

- Pascal is used to describe data precisely; does automatic verification

Structured (Composite) Types

- arrays:
  - lists of items of the same type, e.g.

```
type totals = array[ 1..10 ] of real;
```

  - constituent (base) type may be any type: e.g.

```
array ... of days;
```

  - similar to other programming languages; may construct multi-dimensional arrays
- records: groups of items of different types, e.g.

```
type person = record
                age : integer;
                salary : real;
              end;
```

- can form "arrays of records" e.g.

```
type personnel = array[ 1..100 ] of person;
```

- eliminate need for "parallel arrays": in BASIC, would use one array of integers, another array of reals
- more descriptive of natural arrangement of data
- other types: pointers, used for dynamic data structure applications; sets
- ability to structure data arbitrarily allows programmer to deal with data abstractions -- avoid details of implementation and concentrate on problem

Strings

- standard Pascal: arrays of characters
- no built-in facilities: cannot read, compare different lengths, assign different lengths
- most common extension in Pascal implementations
- WATCOM Pascal allows string manipulation similar to typical BASIC
- standard programs still work correctly

Sequence Control Statements

- two classes: repetition and selection
- repetition:

```
while expression do
    statement;


repeat
    statement;
    statement;
    ...
until expression


for index := initial to final do
    statement;

for index := initial downto final do
    statement;
```

- selection:

```
if expression then
   statement;

if expression then
   statement
else
   statement;
```

```
case expression of
  caselabel : statement;
  caselabel : statement;
  ...
end;
```

- a common extension for the case statement is the "else" case: if all other cases fail, the "else" case is used

Program Structure

- a program consists of a heading, declarations and executable statements

```
program simple( output );
  var i : integer;
  begin
    for i := 1 to 10 do
       writeln( i, i*2 );
  end.
```

- programs may be "broken up" into smaller, isolated pieces: procedures and functions

```
procedure power( base : real, exponent : integer );
  var index : integer;
     result : real;
  begin
    result := 1;
    for index := 1 to exponent do
       result := result * base;
    writeln( result );
  end.
```

- procedures and functions are essential to programming methodology called stepwise refinement

- stepwise refinement: program is divided into many smaller (and usually simpler) components -- each component may be written independently of other components

- functions and procedures have structure similar to a program: heading, declarations, statements

- may be given parameters (values substituted each time); functions return values

Local Declarations

- items declared inside a procedure or function are called local: do not exist until execution of procedure of function begins, disappear when procedure or function finishes
- ability to have local items is useful for data-hiding: allows the separation of the implementation of a procedure or function from its definition
- procedures and functions may contain local types, procedures and functions, in addition to local variables

Requirements for Teaching Software

- typical student (in a classroom):
  - doesn't know what's going on (initially, at least)
  - not interested in writing "production" software
  - is interested in minimizing time to complete task ("get assignment done")
  - needs feedback
- a compiler used for instruction has special requirements
- handle all errors
- diagnose errors in a manner which novices can understand
- errors should be described in terms of Pascal, not machine code
- simple to use
- should not impose restrictions on the language for the sake of run-time efficiency (if feasible)
- must provide fast turn-around -- immediate feedback enhances learning, facilitates experimentation
- should contain facilities which allow programs to be debugged quickly and effectively
- supporting documentation

WATCOM Pascal

- designed for teaching first-year programming courses at University of Waterloo
- interpreter: compile-time is almost zero; program begins execution almost immediately
- integrated with text editor -- enter program and issue single command ("run")
- in simplest form, editor can be learned in about one hour -- no long "learning cycle" before programs can be run
- integrated with debugger
- full ANSI standard language, no restrictions on name lengths, sizes of data structures
- extension where appropriate: graphics, machine-language interfacing, strings
- implemented on a wide variety of machines and operating systems: ICON, IBM PC, Commodore 64, DEC PRO, IBM 370 mainframe (VM/SP CMS), DEC VAX mainframe (VMS)

Debugging with WATCOM Pascal

- debugger is integral part of WATCOM Pascal
- invoked automatically if an error occurs
- user may press "return" to conclude, or issue commands to determine what happened
- debugger may be invoked by pressing "break" (or equivalent) at any time during execution
- standard procedure `pause` can invoke debugger under program control
- when invoked (by whatever method), debugger displays error message and source line
- choice of action: quit; continue (if not an error); execute a Pascal statement; step through execution; redisplay status
- step:
    - allows execution of a single statement each time "return" is pressed
    - may be used to trace program execution
- execute:
    - allows any Pascal statement to be executed
    - may display values, assign values to variables, invoke procedures or functions

An Example

```
program bug( output ); (* sum odd numbers from 1 to 100 *)
  var index : integer;
      total : integer;
  begin
    index := 1;
    total := 0;
    while( index <> 100 )do
      begin
        total := total + i;
        index := index + 2;
      end;
    writeln( total );
  end.
```

- variable i was not declared
- index will never be equal to 100
- program will execute forever!

```
run
Execution begins...

*** Error: 'i' not previously defined
Error executing line 9 in '<main block>'.
      total := total + i;
Debug?
(CR)
```

```
(change i to index)
run
Execution begins...
Break executing line 11 in '<main block>'.
     end;
Debug?
e writeln( index )
       6131
Debug?
w
Break executing line 11 in '<main block>'.
     end;
Debug?
step
     begin (line 8 in '<main block>')
(CR)
       total := total + index; (line 9 in '<main block>')
(CR)
       index := index + 2; (line 10 in '<main block>')
(CR)
     end; (line 11 in '<main block>')
(CR)
     begin (line 8 in '<main block>')
(CR)
       total := total + index; (line 9 in '<main block>')
(CR)
       index := index + 2; (line 10 in '<main block>')
(CR)
     end; (line 11 in '<main block>')
e writeln( index )
       6135
Debug?
e index := 100
Debug?
continue
    9406489

...execution ends
```

```
(change <> to <=)
run
Execution begins...
       2500

...execution ends
```

Support Material

- WATCOM Pascal Primer and Reference
- WATCOM Pascal User's Guide for ...
- WATCOM Pascal for the Commodore 64
- other software:
  - WATCOM Pascal compiler
    - "production" compiler companion to the interpreter -- generate native code for speed and run-time efficiency
    - same language features as interpreter
    - available for ICON, IBM VM/SP CMS, DEC VAX/VMS
  - Waterloo Pascal "load-and-go" compiler
    - generates code and executes directly without need for linkers, etc.
    - very fast compilation rate
    - designed primarily for large-scale timesharing systems: VM/SP CMS, MVS/TSO, VMS, UNIX

Summary

- Pascal is a good language for teaching computer science
- language features which support and encourage good design, modular program construction and data structuring
- language processors used for teaching have special requirements
- teaching software must aid the student as much as possible
- WATCOM Pascal Interpreter is a Pascal processor designed to be used in teaching environments -- responsive, full diagnostics and debugging facilities