

GEM 1501 Problem Solving With Computers

Lecture 3:

Programming Languages, Data Types

Martin Henz

Summary of Previous Lecture

- History of Algorithmics
- Control Structures

Overview of this Lecture

- Programming Languages
- Data Types in Oz

Programming Languages

- Need for programming languages
- Use of programming languages
- Characteristics of programming languages
- Execution of programs
- Collection of languages

Need for Programming Languages

- Programmers are humans
- Humans think in terms of abstractions
- Humans communicate with words and symbols
- Computers are machines that do very simple tasks very fast
- Programming languages bridge the gap between human programmers and computers

Use of Programming Languages

- Programs are precise descriptions of algorithms
- Programs are executed by computers
- Execution involves other programs (compilers, interpreters)
- Compare with training a dog

Characteristics of Programming Languages

- Precise syntax
- Precise semantics

Precise Syntax

- Ambiguities must be avoided
- Examples:
 - `if` statements without `end`
`if true then {Browse 1} else {Browse 2} {Browse 3}`
 - Arithmetic expressions
`1 + 2 * 3`

Syntax Descriptions

- BNF
- Syntax diagrams

Example: For Loops in Oz

```
for I in 0..7 do
  {Browse I}
  {Browse I+100}
end
```

Example: For Loops in Oz

```
 $S ::= \text{for } V \text{ in } E..E \text{ do } S \text{ end}$   
| skip  
| ...  
 $E ::= 1|2|...$ 
```

Precise Semantics: Example 1

```
for I in 0..7 do {Browse I} end
```

Precise Semantics: Example 2

```
for I in 0 .. ~314.1592 do {Browse I} end
```

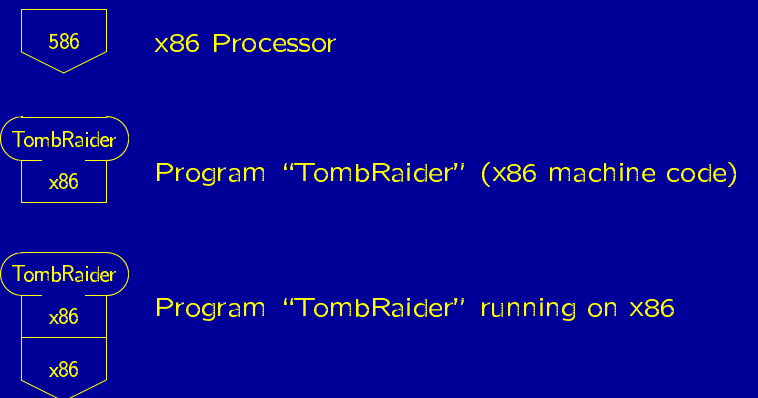
Semantic Issues

- Programmers need to be able to precisely understand each instruction in all its different uses
- Programs need to be re-used in different environments
- Designing programming tools (e.g. debuggers)

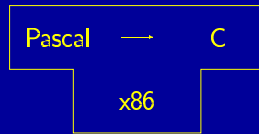
Execution of Programs

- Translators
 - Assemblers; compilers
 - Disassembler; decompilers
- Interpreters
- Combinations (virtual machines)

T-Diagrams

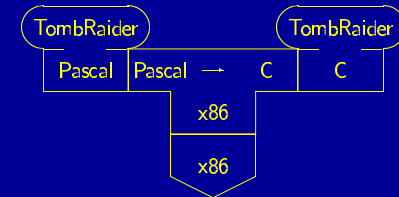


T-Diagram of Compiler



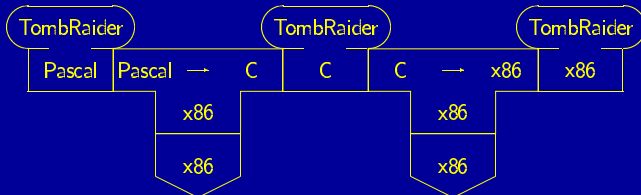
Pascal-to-C compiler in x86 machine code

Compilation



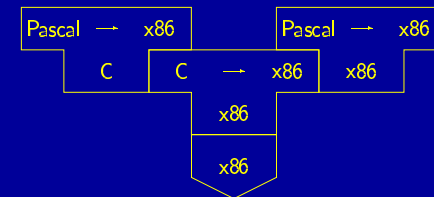
Compiling "TombRaider" from Pascal to C

Two-stage Compilation



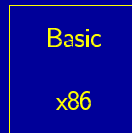
Compiling "TombRaider" from Pascal to C to x86 machine code

Compiling a Compiler



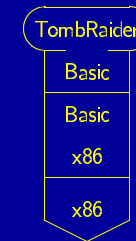
Compiling a Pascal-to-x86 compiler from C to x86 machine code

Interpreters



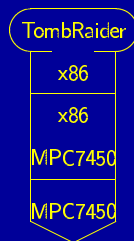
Interpreter for Basic (x86 machine code)

Interpreting a Program



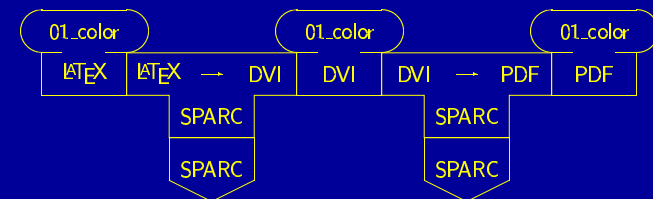
Basic program "TombRaider"
running on x86 using interpretation

Hardware Emulation



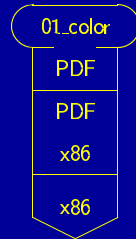
"TombRaider" x86 executable running on a PowerPC
using hardware emulation

Excursion: Making these Slides



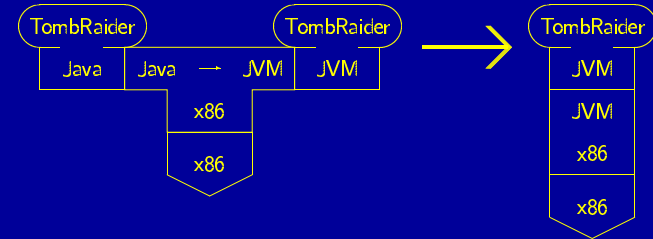
Compiling these slides
from \LaTeX to DVI to PDF on Sun SPARC

Excursion: Viewing these Slides



Viewing the slides on this PC

Typical Execution of Java Programs



Compiling “TombRaider” from Java to JVM code, and running the JVM code on a JVM running on an x86

Summary: Execution of Programs

- Components:
programs, interpreters, compilers, machines
- T-diagrams
- Combination of interpretation and compilation is common
- Interpretation and compilation are ubiquitous in computing

Collection of Languages

- Why not an algorithmic Esperanto?
- FORTRAN
- COBOL
- PASCAL
- SNOBOL
- LISP
- PROLOG
- APL
- OZ

Why Not an Algorithmic Esperanto?

- Why not a linguistic Esperanto?
- Special purpose languages for special applications (XSLT, JavaScript)
- Languages evolving
- Languages die slowly

FORTRAN

- Designed for scientific computation
- Very efficient implementations available
- Great for large array manipulations and other computationally intensive mathematical calculations
- Difficult to design data structures

COBOL

- User-friendly syntax (often looks like English)
- Extensive support for data structures

PASCAL

- Emphasis on "structured programming"
- Procedures can be defined within procedures
- Emphasis on dynamically-created data structures

SNOBOL

- Emphasis on string and text manipulation
- Pioneering language in this area
- Today, replaced by languages such as Perl

LISP

- List-oriented syntax
- Programs are naturally seen as data
- Example: (plus X Y) instead of $X + Y$
- Heavily used in Artificial Intelligence in the 1960s, 70s and 80s

PROLOG

- Variables refer to (possibly partial) information
- Variables can be assigned a value only once
- Close connection to predicate logic
- Example:

```
EATS(cats,mice).  
EATS(cats,frogs).  
EATS(lions,cats).  
EATS(X,Y) :- EATS(X,Z), EATS(Z,Y).
```

```
?- EATS(X,frogs).
```

APL

- Very concise manipulation of arrays and vectors
- Extremely rich set of operators
- Unusual syntax
- Example: $3\ 4\ \rho\ 12$ constructs array

```
1 2 3 4  
5 6 7 8  
9 10 11 12
```

OZ

- Like PROLOG and LISP, single assignments to variables
- Combines functional, object-oriented, and logic programming
- Special support for concurrent, distributed programming
- Special support for combinatorial search

Overview of this Lecture

- Programming Languages
- **Data Types in Oz**

Already Seen

- Integers
- Procedures
- Truth values
- Arrays

New Data Structures

- Atoms
- Records
- Tuples
- Lists
- Characters
- Strings

Atoms in Oz

```
declare
MyAtom = a
```

```
declare
AnotherAtom = z_AA_88_BB
```

```
declare
MyThirdAtom = 'Any sequence of characters!'
```

Records

```
declare
R = a(b:100 c:200 0:300)
```

- the label must be an atom
- the features must be atoms or integers

Tuples

Tuples are records with features from 1 to n where $n \geq 0$

```
declare
T = a(1:100 2:200 3:300)
```

```
declare
T = a(100 200 300)
```

Lists

A list is either the atom `nil` or a tuple with label `'|'` and two components, where the second component is a list.

Examples of Lists

```
declare
L = nil
L1 = '(100 nil)
L2 = '(100 '(200 nil))
L3 = '(100 '(200 '(300 nil)))
```

Abbreviations for Lists

```
declare
L = nil
L1 = 100 | nil
L2 = 100 | 200 | nil
L3 = 100 | 200 | 300 | nil
```

Yet Another Way of Writing Lists

```
declare
L = nil
L1 = [100]
L2 = [100 200]
L3 = [100 200 300]
```

Characters

Characters in Oz are represented by their ASCII code.

```
declare
C = 65
```

```
declare
C = &A
```

Strings

Strings in Oz are lists of characters

```
declare  
S = [65 66 67]
```

```
declare  
S = "ABC"
```

Next Week

- Algorithmic methods