

CSC 4101: Programming Languages

Gerald Baumgartner
gb@csc.lsu.edu

Introduction

Textbook, Chapter 1

2

Why Are We Interested in Programming Languages?

Programming in machine code is way too tedious/error-prone

3

Why So Many Languages?

- **Evolution**
 - From goto to loops, case statements
- **Personal Preference**
 - Syntax
 - Loops vs. recursion
 - Pointers vs. recursive data types
- **Special Purposes**

7

Application Domains

- **Scientific applications (Fortran, TCE)**
- **Business applications (Cobol)**
- **Artificial intelligence (Lisp)**
- **Systems programming (C, C++)**
- **Web service programming (Java, C#)**
- **Very High-Level Languages (perl)**
- **Special purpose languages (make, sh)**

8

What Makes a Language Succeed?

- **Expressive Power**
- **Ease of Use for Novice**
- **Ease of Implementation**
- **Open Source**
- **Availability of Compilers, Libraries**
- **Economics, Patronage, Inertia**
- **Syntax that looks like C**

9

Language Design Issues

- Readability (*p++)
- Abstractions (functions, classes)
- Orthogonality (no special cases)
- Reliability (type checking)
- Cost (training programmers)

10

Programming Paradigms

- Imperative (C, Pascal, etc.)
- Functional (Lisp, ML, Haskell)
- Logic (Prolog)
- Object-Oriented (C++, Java, CLOS)

11

Why Do We Study Programming Languages?

- Understand obscure language features
- Choose among ways to express ideas
- Make good use of debuggers, other tools
- Simulate nice features in other languages
- Choose appropriate language for problem
- Learn new languages faster
- Design simple languages

12

Implementation Methods

- **Compilation (C, ML)**



- **Interpretation (early Lisp)**

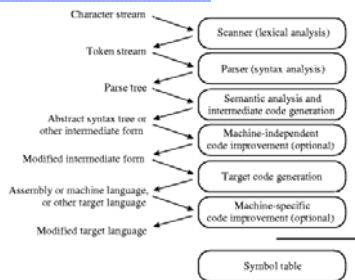


- **Hybrid Systems (early Java)**



13

Overview of Compilation



14

Source Code for GCD

```
program gcd(input, output);
var i, j: integer;
begin
  read(i, j);
  while i <> j do
    if i > j then i := i - j
    else j := j - i
  writeln(j);
end.
```

15

Tokens (After Lexical Analysis)

```
PROGRAM, (IDENT, "gcd"), LPAREN,
  (IDENT, "input"), COMMA,
  (IDENT, "output"), SEM,
VAR, (IDENT, "i"), COMMA,
  (IDENT, "j"), COLON, INTEGER, SEM,
BEGIN
...

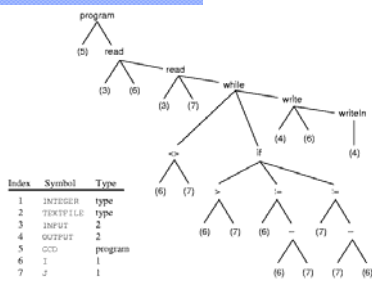
```

16

Parse Tree



Syntax Tree and Symbol Table



18

Assembly

```
addiu $p,$p,-32 # reserve room for local variables
sw $ra,20($p) # save return address
jal getint # read
nop
sw $0,20($p) # store i
jal getint # read
nop
sw $0,20($p) # store j
lw $t0,20($p) # load i
lw $t1,24($p) # load j
nop
bne $t0,$t1 # branch if i != j
nop
A: lw $t0,20($p) # load i
lw $t1,24($p) # load j
nop
slt $t2,$t0,$t1 # determine whether j < i
bne $t2,$t2,$t2 # branch if not
nop
lw $t0,20($p) # load i
lw $t1,24($p) # load j
nop
subu $t2,$t0,$t1 # i2 := i - j
sw $t2,20($p) # store i
lw $t0,20($p) # load i
lw $t1,24($p) # load j
nop
subu $t2,$t1,$t0 # i5 := j - i
sw $t2,24($p) # store j
C: lw $t0,20($p) # load i
lw $t1,24($p) # load j
nop
bne $t0,$t1 # branch if i <> j
nop
D: lw $0,20($p) # load i
jal print # write
nop
sw $0,$zero # exit status for program
nop
b E # branch to E
nop
E: lw $ra,20($p) # retrieve return address
addiu $p,$p,32 # deallocate space for local variables
jr $ra # return to opening system
nop
```
