Index

ABC, programming language, compiler for “Elbrus,” 14
Action, semantic, 11
ActionScript, 48, 92
Ada, programming language, implementation for “Elbrus,” 15
Addressing
   variables of different types, 181–186
   uplevel, 19, 157
Address, pair, 189
   level of, 189
   offset of, 189
Algebra, formal structure
   carrier of, 23
   equate(s) in, 23
   multi-sorted, 23
   signature of, 23
   sort of, 23
ALGOL, 13. See also ALGOL-60
ALGOL-60, 13–14, 32
   spelling of keywords in dialects of, 32
   use of display in the implementation, 116, 189
ALGOL-68
   formal definition by Van Wijngaarden
      two-level grammars, 54
   mode (type) definitions in, 104, 120
   Soviet implementation for IBM-360, 14
   structural type identity in, 121
   user-defined keywords and indicators, 40, 44
ALPHARD, 18
Analysis
   control flow, 131, 158, 162, 246, 260
   data flow, 157, 162, 249, 260, 261
ANTLR, compiler development tool, 2, 11, 45
   input grammar file structure, 48–49
   lexical analysis specification and generation, 48–51
   lookahead buffer, 49
   nextToken method, 50
   parser specification and generation, 92–93
ArrayIndexOutOfBoundsException, 32, 173
as, assembler, 180
Aspect-oriented programming (AOP), possible application in compilers, 58
Aspect.NET, 27, 240–241, 246, 249
Assertion(s), 3, 21
Assignment, static single (SSA), 163–164 in Phoenix, 262–264
phi function in, 164, 264
Attribute, semantic, 11, 52, 99. See also Grammars, attributed
dependency graph (DAG), 100
evaluation in Sun Pascal, 111–113
evaluating on the parse tree, 100–101
of expression, 110
global, 101
inherited, 99
of type, 109–109
of variable, 109–110
synthesized, 99–101
Automaton(-a)
finite-state, 35, 37
stack-based, 54
Back-end, 8
BASIC, 13, 14
compiler for “Elbrus,” 13
bison, compiler compiler, 11, 81, 83, 87–89
generalized parsing in, 87–89
resolving conflicts, 88
semantic actions, 88–89
Block
basic, 127, 132, 150, 154–157, 160, 162, 163, 165, 173, 260, 261, 262
structure, 19
Bootstrapping, 10–11
Borland JBuilder, 6, 221
C, 1, 4, 15, 28, 32, 59, 83
implementation for “Elbrus,” 15
pointer arithmetic, 118
type casts in, 118
type definitions in, 120
verifying compiler, 18, 29, See also vcc
C++, 1, 4, 6, 9, 11, 26, 28, 29, 32, 84, 89, 92, 93, 165, 201, 202
container variable model in, 181
inline pragma(s) in, 34, 161
relaxed typing in, 133
user-defined memory allocation in,
unmanaged pointer(s) in, 185, 187
C#, 1–2, 5, 6, 7, 25, 26, 31, 32, 43, 48, 55, 83, 92, 114, 117, 119, 121, 130, 161,
static identification in, 114
strong typing in, 119
use of the same identifier as a field and a method name, 117
Caml, 25–26
Objective, 26
Car, list operation, 23–24
Cdr, list operation, 23–24
Chain
dynamic (list of called functions), 114, 188
static (list of active scopes), 188
CIL. See .NET, Common Intermediate Language
CLU, 13
efficient algorithm of type handling in, 123–126
equates in, 103–104
implementation for “Elbrus,” 14, 97
procedure type (proctype), 124
structural type identity in, 121
COBOL, 5, 14
implementation for “Elbrus,” 15
Code
generation, 7, 8
for assignment statements, 197
for automatic (local) variables, 197
for case (switch) statements, 198–199
for expressions, 197–198
for if statements, 198
for global declarations, 197
for .NET, 179–180
tasks and goals of, 175–179
generator, 8
intermediate, generation of, 98, 141–142
native, 5, 6, 8
object, 1, 8
resilience to attacks, 19
trustworthy generation of, 19
trustworthy, 2
source, 1, 7
virtual, 8
CoCo/R, compiler development tool, 2, 11, 83, 93–95
Cohesion, temporal, 9
Comments, control, 34
Compiler(s), 1
ahead-of-time (AOT) 2, 5, 200, 220–221, 242. See also pre-compilers
binary, 5
compiler(s), 11
development
boom for .NET, 2
history of, 13–15
driver, 20, 141, 166, 180, 239
GNU, 20, 142
graph, 5, 222–238
generation tools for, 237–238
incremental, 4, 20
interpreter (driver) of steps, 4
step, 4
just-in-time (JIT), 2, 4, 203–220
architecture of, 204–205
history of, 170–171
for SSCLI/Rotor, FJIT compiler, 211–212
for SSCLI/Rotor, optimizing JIT compiler, 212–220
kinds of, 4–5
multi-pass, 9
one-pass, 9
pass(es) of, 9
phases of, 7–8
Sun Pascal, 13, 62, 71, 83, 97, 111, 113, 114, 117, 131, 132
refurbishment, 13, 113, 114
Sun Studio, 9, 33
acomp, C compiler front-end, 166, 180
cg, code generator, 166, 180
IR, 142, 165
iropt, IR optimizer, 166, 180
optimizations in, 165–167
organization of, 166
system(s)
ALPHA, 14
BETA, 14
TA-1M, ALGOL-60 Soviet compiler, 13–14
traditional, 5
trustworthiness
of data, 20
of the implementation code, 20–21
trustworthy, 1, 2–4, 17–22
trustworthy behavior of, 19–20
trustworthy error diagnostics and recovery in, 21
verified, 22, 23
Leroy’s approach, 24–26
verifying, 3, 17–18, 21–22
Computation(s)
mixed, 152, 154–155
concretization, 154
implementation for .NET, 154–155
residual data in, 154
residual program in, 154
Computing, trustworthy, 2
initiative (TWC), pillars of 16–17
Cons, list operation, 23–24
Context-dependent analysis, see Semantic analysis
Converter(s), 5
Cross-compiler(s), 4
Current Effective Definition (CED), 39, 101–102
Data, misaligned, 108, 182–183
Debugging, information, 179
generation of (in SPARC), 195–196
ILDB (debugging information format in Rotor), 195
PDB (debugging information format in .NET), 195
STABS (debugging information format in SPARC), 195–196
Definition
Current Effective (CED), 39, 101–102
of an identifier, 8
regular, 37
system(s), 103–104
algorithm D for one-pass analysis, 104–107
dependency graph G(D) of, 104
dummy entry in, 103, 117–118
table of, 103, 115, 122–123
entry structure, 115–117, 122
DELTA, compiler compiler, 11
Dependency(-ies)
control flow, 165
antidependent statement, 162
control dependent statement, 162
input dependent statement, 163
output dependent statement, 162
data flow, 162
data flow dependent statement, 162

Descent, recursive, 9, 52, 55–61
conditions to be applicable, 57
eat parsing procedure, 56
error recovery in, 62–68
intelligent panic mode, 65–66
lookahead mechanism, 59–61
panic mode, 62
parsing procedures Pr(A), 56–58
skipping until a pair of synchronizing
tokens, 66–67

Design by contract, 3, 7, 18, 26, 29
ensure statement, 26
object invariant, 26
require statement, 26

Display
use at runtime, 189
use with the table of definitions,
115–116

Eiffel, programming language, 29
“Elbrus,” Soviet computers, 12,
compilers for, 13, 14, 15, 97, 106
expensive memory requests to OS, 203
hardware implementation of display
and procedures, 189
interoperability of languages, 202
tag-based runtime checks, 187, 202
using postfix notation, 165
EL-76, programming language, 15
Error, diagnostics and recovery, 3, 7–8
Exception, formalization of, 24
Execution, managed, 6, 201–202, 221
Expression(s), regular, 35–37
Equate(s), 23, 24, 103, 104

F#, 26
Form, Extended Backus-Naur (EBNF),
53, 54, 56, 90, 269
FORTH, 4, 20
implementation for “Elbrus,” 13, 14
unsafe programming style, 145

FORTRAN, 9, 13, 14, 19, 165
implementation for “Elbrus,” 15
lexical analysis issues, 44–45
linearization of arrays column-by-
column, 184
types, 118
Front-end, 8–9

File, object, format(s) of
Code Object File Format (COFF),
179, 246
Executable and Linkage Format
(ELF), 179
Portable Executable (PE), in .NET,
179

Grammar(s)
attributed, 11, 99–103
automata, 35, 37
context-free (CF), 7, 11, 35, 53
graph, 222–238
definition of, 226
domains of application, 223
hyper-graph approach, 235–237
layered, 232–233
meta-modeling approach, 233–235
reserved, 230–232
language of, 36
L-attributed, 102
Bochmann’s condition, 102
right-linear: see automata grammar

Graph(s)
control flow, 162, 165, 260
data flow, 162, 165, 261,
formal definition(s), 224
labeled, 224
morphism, 224–225
Pacman, example, 227–230
transformation(s), categorical
approach, 226–230

grep, 37

Hardware, architecture(s)
Burroughs
hardware implementation of display
and procedures, 189
hardware interoperability of
languages, 202
hardware tag-based runtime checks,
186, 202
CISC, 189, 190
Explicit Parallelism Instruction Computer (EPIC), 1, 277
multi-core, 1, 87, 174, 175, 277
M-20, Soviet computer, 13
M-220, Soviet computer, 13
M-222, Soviet computer, 13
RISC, 150, 174, 175, 176, 178
Very Long Instruction Word (VLIW), 1, 174, 175, 178, 277
x64, 1, 4, 8
x86, 1, 4, 8
Hash function(s), 39–40, 41
use of type sort as, 122
Identifier, table
id_tab, 33, 38–42, 102
id_nl field, 38, 115–117. See also Current Effective Definition
ild, incremental linkage editor, 180
Inheritance, 119, 129
Inlining, See Optimization(s)
Interpreter(s), 4, 5, 6, 13, 14, 15, 145
Intrinsic(s), 108
Invariant
abstract, 18
concrete, 18
Java, 1, 4
bytecode, 4, 5, 6, 8, 91, 142, 145, 146
compilers, evolution of, 5–6
HotSpot, 6, 168, 171
static identification in, 114
strong typing in, 119
virtual machine (JVM), 5
JavaCC, parser generator, 11, 83, 90
JavaScript, 2, 28, 48, 92, 119. See also ActionScript
Kleene’s theorem, 37
Knowledge.NET, 5
Language(s)
high-level, 1
LL, 68
LR, 68
ld, linkage editor, 180
Level
nesting, of the block: see Static level static, of the block, 116. See also Nesting level
lex, generator of lexical analyzers, 35, 45–48
example of specification of lexical analyzer, 46–47
library, 48
output, lex.yy.c file, 48
prototype of generated lexical analyzer, 45
rules, 45–46
semantic actions, 46
structure of an input file, 46
user-defined routines, 46
variables, 47
Lexical analysis, 7, 8, 31–51
ambiguity issues, 44–45
auxiliary functionality, 34
error diagnostics and recovery, 38
processing identifiers and keywords, 38–42
efficient algorithm of, 41–42
processing white spaces and comments, 34–35
theoretical models of, 35–38
Lexical analyzer
architecture of, 42–43
global interface variables of, 33–34
output of, 33–34
principles of trustworthy implementation, 43–44
and scanner, 37–38
Link, static, of a procedure, 187–188
Linking, 180
lint, static code checker, 131, 132, 133
Linux, 9, 20, 165
LISP, 4
Cons, Car and Cdr operations in, 23–24
dynamic identification in, 114
implementation for “Elbrus,” 14
weak typing in, 119
Lock, synchronization object, 19
Lookahead, 36, 38, 44, 47
token buffer, 44, 50, 59
Lookup, 8, 98, 113–118
efficient method, 115–117. See also Current Effective Definition (CED)
lookup (cont’d)
   application in Sun Pascal, 117
   modification for object-oriented
   languages, 117
   error diagnostics and recovery,
   117–118
   traditional methods, complexity of 114

method(s), representation at runtime,
   187
   virtual method table (VMT), 187
 ml, 25
 mobile device(s), rapid development of, 2
 modula-2, 93
   implementation for “Elbrus,” 13, 14
 module(s), as part(s) of intermediate
   representation (IR), 150
 monitor, synchronization concept, 19
 msil. see .net, microsoft intermediate
   language
   multi-threaded (MT) safe, 19
 mutex, 19

namelist. see table(s), of definitions
 .net, 1, 2
 common intermediate language
   (cil), 8, 27, 98, 145. see also
   microsoft
   intermediate language
 common language runtime (clr),
   6, 19, 119, 185, 201, 205
 common type system (cts), 179,
   202, 257
 compilation for, 6–7
 dynamic language runtime (dlr),
   119, 129
 managed execution, 6
 metadata, 6, 27, 185, 202, 204, 205,
   212, 214, 215, 218, 219, 253, 257
 microsoft intermediate language
   (msil), 6. see also cil
 multi-language programming, 2, 179
 ngen utility, 96, 220
 regular expression analysis APIs, 43
 nl. see table(s), of definitions
 notation, postfix (reverse polish), 98,
   142–146
   algorithm of translating expressions
to, 143–145
 occurrence (of an identifier)
   applied, 8, 101, 114
   list of, 9, 103, 104, 107, 113, 151
   defining, 101, 114. see also definition
 optimization(s), 7, 8, 141, 152–173
   arithmetic simplifications, 156
   common subexpression elimination,
   156–157
   constant folding, 153, 155
   constant propagation, 155
   copy propagation, 155–156
   dead code elimination, 157–158
   erroneous (deleting uplevel variable
   assignments), 157–158
   during JIT compilation, 170–173
     basic block, 173
     constant folding, 210
     dead code elimination, 211
     deoptimization (of cold methods),
     211
     devirtualization, 172, 210, 214, 215,
     216
   elimination of redundant
   assignments, 205, 207
   elimination of redundant runtime
   checks, 173
   example of optimizations for .NET
   and Rotor, 205–210
   inlining, 171–172
   inlining of hot methods, 205
   inlining of short methods, 210
   loop invariant code motion, 211
   selective (based on profiling), 171
   global, 154, 167
   inlining, 34, 161–162
     vs. encapsulation principles, 171
   of Java bytecode, 167–170
     of constant pool, 168–169
     -O option of the Java compiler, 169
     pattern-based peephole, 169
   in Microsoft .net, 154, 170
   as mixed computations, 154–155
   in Sun Studio, 165–167
   local, 154
   loop
     induction variable elimination,
     159–160
     invariant code motion, 158–159
     loop unrolling, 160–161
“over-optimization,” 153
peephole, 154
pessimization: see “over-optimization”
platform-dependent, 153
platform-independent, 153
strength reduction: see arithmetic simplifications
trustworthiness of, 153
unreachable code elimination, 158
Optimizer, 8, 11, 141

Parse tree, 7, 8, 52, 54
Parsing, 7–8, 52–96
basic concepts and principles, 53–55
bottom-up (shift-reduce) 54–55, 63, 67–71
accept action, 69
error action, 69
handle(s), 68
reduce action, 69
reduce-reduce conflicts, 71–72
shift action, 69
shift-reduce conflicts, 71
deterministic, 54
error recovery in, 62–67
generalized, 54, 87–89
LALR(1), 55, 80–81, 83
LL(1), 55
of definition system, 103
LR(1), 55, 77–80
canonical collection of sets of LR(1) items, 78–79
closure function, 77–78
error recovery in, 81–82
goto function, 78
LR(1) items, 77
parsing table(s), 79–80
nondeterministic, 54
SLR, 72–76
canonical collection of sets of LR(0) items, 75
closure function in, 73
goto function in, 74
LR(0) item(s), 72–73
parsing tables, 75–76
top-down, 54–55
Pascal, programming language, 1, 4, 5, 9, 10, 13, 26, 31
approach to keywords, 32, 40
control comments, 34
implementation for “Elbrus,” 14, 15
processor, 127
strong typing, 118
Turbo Pascal compiler, 21
type(s), 118
assignment compatibility, 127
compatibility, 127
definition(s), 119–120
enumeration(s), 118
file(s), 118
identity, 120, 126
record(s), 118
set(s), 118
P-code, 145
Perl, 2, 28
Phoenix, compiler development kit, 9, 11, 142, 164, 170, 239–276
alias analysis, 264
architecture, 242–246
c2, C++ compiler backend, 247, 250, 251, 261, 263, 265, 267
collection(s), 246, 248
control(s), 248
data flow analysis, 260–262
exception handling, 265
extension object(s), 264
graphs, 262
call, 262
flow, 262
inheritance, 262
interference, 262
history of, 240–242
intermediate representation (IR), 165, 170, 248–253
encoded (EIR), 249
high-level (HIR), 243, 248–253
low-level (LIR), 243, 251–252
mid-level (MIR), 243, 249
name(s), 248
Phoenix-FETE project, 95, 241, 243, 267–274
Phoenix HL-AST project, 241
plug-in(s), 244, 246–247, 265–267
example of, 265–267
reader(s), 245, 246
safety, 264
simulation-based optimization(s), 265
static analysis, 264–265
Phoenix, compiler development kit (cont’d)
empty catch blocks, 265
hidden members of types, 265
memory leak detection, 265
non-initialized variables, 265
security validation, 265
unnecessary object creation, 265
unused variables, 265
static single assignment (SSA), 262–264
string(s), 247–248
symbol(s), 253–257
table(s), 253–257
table map(s), 253
tool(s), 170, 246, 247, 267
type system, 257–260
use at St. Petersburg University, 240–242
writer(s), 246
piVC, Pi language verifying compiler, 18
implementation for “Elbrus,” 15
type definitions in, 120
Pragma(s), 7, 34, 161, 167
Precondition, 18, 22, 26, 28
Preprocessing, 180
Privacy, of the compiler data, 3
Postcondition, 18, 22, 26, 28
Procedure(s), representation at runtime, 186–188
PROLOG, implementation for “Elbrus,” 15
Python, 2, 28, 92, 119, 129
Record, activation, in a stack, 25, 150, 181, 187–189, 191, 197, 200
Reengineering, 5
REFAL, implementation for “Elbrus,” 13, 14
Representation, intermediate (IR), 8, 19, 20, 82, 98, 141–150, 175
architecture models, 142
requirements to, 141–142
Ruby, 2, 119, 129
Runtime, 200–203
garbage collection, 201
copying, 201
conservative, 201
generational, 201
mark-and-sweep, 201
interoperability of languages, 202
in .NET, 202
in Java and .NET, 201. See also
Execution, managed
memory management, 200
relationship with the operating system (OS), 202–203
exception handling, 203
memory management, 200
threading, 201
security checks at, 201
stack and procedures management at, 187–189, 200
threads management at, 201
type-checking, 201–202
array index out of bounds checks, 201
 subrange checks, 201
SableCC, parser generator, 2, 11, 83, 90–92
Scanner. See under Lexical analyzer
Scope, 39, 101, 103, 114–118
different name “universes,” 117
Security, 16–17, 19, 118, 121, 185, 186
check(s), 20, 201
Semantics, mathematical models of, 22
abstract state machine (ASM), 23
algebraic, 22, 23–24
denotational, 22, 23
Hoare’s program calculus (triples), 22
operational, 22–25
Semantic analysis, 3, 8, 9, 11, 17, 44, 72, 97–151
checks for context restrictions in Sun
Pascal, 139–141
design flaws of, 111
error diagnostics and recovery in, 98
formal model, 99. See also Grammars,
resolvers, 99
visitors, 99
trustworthiness checks, 131–139
array index out of bounds, 132
dangerous use of weak typing
features (in C and C++), 133
defining variables that are never used, 132
“fall through” in switch statements, 132–133
incorrect string comparison (in Java), 133
synchronization bugs (potential deadlocks), 133–134
unreachable logical conditions, 132
using non-initialized record fields (our method), 134–139
using non-initialized variables, 131–132
signal, synchronization operation, 19
SIMULA-67, implementation for “Elbrus,” 15
SNOBOL, 4
dynamic identification, 114
implementation for “Elbrus,” 13, 14
weak typing, 119
Solaris, 9, 20, 165, 174, 179, 180, 203
SPARC, hardware architecture, 1, 8, 9, 15, 165, 166, 170, 174
Application Binary Interface (ABI) conventions, 191–192
architectural principles, 190–192
assembler
section(s), 194
directive(s), 195
code generation example for, 192–194
frame pointer register (%fp), 191
register pool, 191
register window(s), 191–192
registers, in, local and out, 191
representation of local variables on the stack, 191–192
stack pointer register (%sp), 191
Spec#, 7, 18, 24, 26–28, 29
Boogie verifier, 27
modifies specification, 28
old keyword, 28
SUPER, compiler compiler, 11, 14
Symbol(s)
nonterminal, 35, 53
terminal, 35, 36, 53
undefined, processing of, 103
Syntax analysis, 7. See also Parsing
Table(s)
of definitions, 103, 115, 122–123
dummy entry in, 103, 117–118
entry structure, 115–117, 122
of types, 121–123
TIP technology, 12–13, 15, 21, 58, 148–149
abstraction layer(s), 12
access interface, 12
concrete representation, 12
conceptual layer, 12
definition layer, 12
generation interface, 12
output interface, 12
representation layer, 12
technological instrumental package (TIP), 12
TGenField, sample operation, 13
TGetField, sample operation, 13
TGetMem, sample operation, 13
update interface, 12
vertical cut(s), 12
Token(s), 7
classes, 31–33
delimiter, 33
identifier, 32
keyword(s), 32
number denotation, 32
operator symbol, 32
string denotation, 32
symbol (character) denotation, 32
stream of, 7, 31, 34, 52
synchronizing (in error recovery), 62
as terminal symbols in parsing, 53–54
Translation, syntax-directed, 83
Translator(s), 13, 14. See also Compiler(s)
Tree(s)
abstract syntax (AST), 48, 52, 53, 98, 100, 111, 237
attributed, 111
derivation, 7, 52. See also parse
parse, 7, 8, 52, 54
terminal nodes (leaves), 54
PCC (Portable C Compiler) tree(s), 82, 98, 142
Tree(s) (cont’d)
  architecture, 146–149
  implementation in C, 147–148
  type encoding in (TWORD), 147
Triple(s), 98, 142, 149–150. See also
  Representation, intermediate
    leaves, 150
    operation code, 149
    numbering, 149–150
    segment(s), 150
    types, 149
    use in Sun Studio compilers, 150
Type(s)
  abstract data (ADT), 12, 18, 21, 23,
    118, 130, 161
  alias(es), 120
  array, 120
    compile-time representation of,
      122–123
    dope vector: see passport
    elements type, compile-time
      representation of, 123
    index type, compile-time
      representation of, 123
    passport, 108, 159, 160, 185
    assignment compatibility, 110, 126–127
    checking, 8, 98, 128–131. See also
      Typing
        for expressions, 128
        for generic types, 130–131
        for indexed variables, 129–130
        for record and object members, 129
        compatibility, 127
        definition(s), 119–120
        denotation(s), 120, 121, 122, 123
        error recovery, 128
        expression, 120. See also denotation
        evaluation: see checking
      generic, 130–131
        in C#, 130
        in CLU, 130
        concretization(s): see instantiations
        instantiation(s), 130
        in Java, 130
        type identity for, 130
      identifier, 119–120
      identity, 120
      name, 120
      in object-oriented languages, 120
structural, 120
structural, efficient method and
  algorithm to handle, 123–126
structural, insecurity issues of, 121
structural, module interoperability
  issues of, 121
parametrized: see generic
  principles of efficient, 121–122
  representation at compile time,
    121–123. See also Table(s),
    of types
  use of the id_tab, NL and Types
    tables to represent a type, 122
representation at runtime, 180–186
  arrays, 184–185
  compound types, 181
  pointers, 185–186
  primitive types, 181
  record types (structures), 182–184
  variant records (unions), 183–184
subrange, 123
  lower bound, 123
  upper bound, 123
synonym(s), 120
  table of, 121–123
Typing. See also Type(s), checking
  dynamic, 119, 129
  strong, 119
  weak, 119
UNIX, 20, 37, 45, 46, 49, 81, 83, 166
UML (Unified Modeling Language)
  activity diagram(s), 19, 238
  diagram(s), 233, 234
Variable
  input, of a statement, 165
  output, of a statement, 165
vcc, verifying C compiler, 18, 29
Verifier(s), formal, 18, 29
  in Spec#, 7, 27, 28
Virtual machine, 4, 8, 13
Visual Basic.NET, 2, 6, 203
Visual Studio, 26, 27, 28, 170, 221, 240,
  242, 245, 267, 268, 271, 272
wait, synchronization operation, 19
Web programming languages, dynamic
  typing features of, 2, 119
x64. See under Hardware architecture(s)
x86. See under Hardware architecture(s)

yacc, 11, 31, 33, 37, 45, 46, 55, 63, 71, 81, 82–87, 101
command-line interface, 86
token definition(s), 85
example of input grammar, 84–85
input and output, 83
input file structure, 84
precedence of operations, 85
semantic action(s), 86
semantic stack, 86
syntax stack, 86
use in Sun Pascal, 111–112
yacc++, parser generator, 89–90