

Contents

Ada Scanner and Parser	1
Supported Subset	1
Requirements	2
Build	2
Run	2
Usage	2
Directory Tree	3
Grammar	3
Operator Precedence	3
AST	4
Output	4
Memory Management	4
Examples	4
Authors	4
Acknowledgments	4

Ada Scanner and Parser

A scanner (Flex) and parser (Bison) for a small, well-defined subset of Ada. The parser builds an Abstract Syntax Tree (AST) for a single main procedure and supports pretty-printing and a structural debug dump of the AST.



Supported Subset

- One procedure: `procedure Main is ... end Main;`, but is made to read any name of procedure, not just `Main`;
- Statements:
 - Assignment: `x := expr;`
 - If / If-Else: `if expr then ... [else ...] end if;`
 - While: `while expr loop ... end loop;`
 - I/O: `Put_Line(expr);` and `Get_Line(id);`
- Expressions:
 - Integers and floats;
 - Strings with Ada doubled-quote escaping (`"He said ""hi"""`);
 - Identifiers
 - Operators:
 - * Arithmetic: `+` `-` `*` `/` `**` `mod` `rem`
 - * Boolean: `and` `or` `xor` `not`

- * Comparisons: = /= < > <= >=
- * Parentheses: (expr)
- Lexical:
 - * Line comments: -- ... (until end of file)
 - * Whitespace ignored

Requirements

- Build tools: `make`, `flex`, `bison` and `gcc`
- Notes:
 - Link with `-lm` for portability
 - `-lfl` is not required since lexer uses `%option noyywrap`

Build

- With Makefile (recommended):
 - `make && mv printAST printAST_{your OS}`
 - Regenerates `parser.tab.c/.h` and `lex.yy.c` when needed and builds `printAST_{your OS}`
- Manual (fallback):
 - bison: `bison -d parser.y`
 - flex: `flex lexer.x`
 - gcc `-Wall -g -o (nameOfExecutable) parser.tab.c lex.yy.c ast.c ast_debug.c main.c -lm`

Run

There are two different executables in this repository: `printAST_Mac` and `printAST_Linux`. The user must choose the one with its OS. If the user wants to create another one, go to Build

- Pretty-print (source-like):
 - `./printAST_{your OS}`
 - or `./printAST_{your OS} < path/to/file.ada`
- Debug AST dump (structural):
 - `./printAST_{your OS} --debug path/to/file.ada`

Usage

- `./printAST_{your OS} [--debug] [file]`
 - `--debug` prints a structural AST dump
 - `file` is optional; if omitted, reads from `stdin`
- Exit status: non-zero on lexical/syntax errors

Directory Tree

```
Ada_Compiler/  
├── code/  
│   ├── ast_debug.c  
│   ├── ast_debug.o  
│   ├── ast.c  
│   ├── ast.h  
│   ├── lexer.x  
│   ├── main.c  
│   ├── Makefile  
│   ├── parser.y  
│   ├── printAST_Mac  
│   ├── printAST_Linux  
│   └── test_inputs/  
│       ├── input1.ada  
│       ├── input2.ada  
│       ├── input3.ada  
│       └── input4.ada  
├── worksheet/  
│   └── Compiladores_Trabalho.pdf  
└── README.md
```

README.md

Grammar

- Program: procedure ID is begin stm_list end ID;
- Statement list: zero or more stm
- Statements:
 - ID := expr;
 - if expr then stm_list end if;
 - if expr then stm_list else stm_list end if;
 - while expr loop stm_list end loop;
 - Put_Line(expr);
 - Get_Line(id);
- Expressions: literals | ID | unary - / not | binary ops | (expr)

Operator Precedence

From highest to lowest:

1. Unary: not, unary -
2. Power: ** (right-associative)
3. Multiplicative: * / mod rem
4. Additive: + -
5. Relational/equality: = /= < > <= >=
6. Boolean AND
7. Boolean OR / XOR

AST

- Expressions: NUM, FLOAT, ID, STRING, BOOL, OP(op, left, right), UNARY(op, child)
- Statements: ASSIGN(id, expr), PUT_LINE(expr), GET_LINE(id), IF(cond, then, else), WHILE(cond, body), COMPOUND(left, right), PROC(name, body)
- Printing modes:
 - Pretty (Ada-like) via `print_stm`
 - Debug (Structural) via `debug_print`

Output

Using `test_inputs/input3.ada`:

- Pretty:

```
PROCEDURE Main IS BEGIN x := 5 ; IF x > 3 THEN PUT_LINE("ok"); END IF; END Main;
```

- Debug:

```
PROCEDURE(Main)
body:
  ASSIGN(x, NUM(5))
  IF
    cond: OP(GT, ID(x), NUM(3))
  then:
    PUT_LINE(STRING("ok"))
  else:
    (none)
```

Memory Management

- Nodes and strings allocated with `malloc` / `strdup`
- Entire AST freed with `free_stm(root)` which recursively frees expressions/statements and owned strings

Examples

See `test-inputs/`.

Authors

Made by Maximiliano Sá and Rita Moreira.

Built as part of coursework at Faculdade de Ciências da Universidade do Porto.

Acknowledgments

- ada-auth.org
- ada-lang.io