Bootstrap Ada 2022 Compiler

Project Goal

To build a self-hosted Ada 2022 (ISO/IEC 8652:2023) compiler written entirely in Ada, bootstrapped from a validated Ada 83 foundation, targeting Intel 80486 architecture. The final compiler is written entirely in Ada 2022 and is capable of compiling itself.

Bootstrap Strategy

A self-hosting compiler cannot compile itself before it exists. The project uses a four-stage bootstrap path. The C-language Ada/Ed platform is used at Stage 0 only as a development tool. From Stage 1 onward the compiler is written entirely in Ada with no C source code:

Stage	Description	Language	Status	Work Hours with Claude
0	Validated Ada 83 platform	C (NYU Ada/Ed)	**Complete**	7
1	Ada 83 compiler front-end	Ada 83	**Complete**	9
2	Self-hosting Ada 83 compiler	Ada 83	**In Progress**	—
3	Ada 95 extensions	Ada 83/95	Planned	—
4	Ada 2022 (ISO/IEC 8652:2023) extensions	Ada 95/2022	Planned	—
5	Ada Compiler Acceptance Test Suite (ACATS) validation	—	Planned	—

Stage 0 — Validated Ada 83 Bootstrap Platform

Completed: May 15, 2026 | Implementation Language: C

The foundation is NYU Ada/Ed version 1.11.2 — the first validated Ada compiler ever certified (ACVC 1.7, April 11, 1983), and the first validated Ada implementation for the IBM PC. Ada/Ed is written in C and is used as the bootstrap platform during development only. It is not part of the final compiler. Stage 0 establishes an Ada 83 compilation platform implemented in C. Writing the compiler itself in Ada begins in Stage 1 and all subsequent stages contain Ada source code only, with no C whatsoever.

Source Repository

```
github.com/softwarepreservationgroup/adaed
```

Accomplishments

- Downloaded complete NYU Ada/Ed 1.11.2 C source — 241 files, 3 MB
- Compiled all 113 C source modules to Intel 80486 32-bit assembly using TDM-GCC 10.3.0 on 64-bit Windows
- Resolved all platform compatibility issues between the 1996 DOS/Intel C Code Builder environment and modern Windows
- Built all five Ada/Ed executables entirely from C source — no pre-built binaries used
- Rebuilt Ada 83 standard library (predef.*) from Ada 83 source code
- Compiled and executed the first Ada 83 program successfully

Executables Built

Executable	Size (bytes)	Function
`adafront.exe`	957,712	Parser and semantic analyzer
`adagen.exe`	770,060	Ada machine code generator
`adabind.exe`	647,004	Compilation unit binder
`adaexec.exe`	409,535	Ada machine interpreter/executor
`adacomp.exe`	454,023	Master compiler driver

Proof of Success

```
with TEXT_IO; use TEXT_IO;
procedure HELLO is
begin
   PUT_LINE ("Hello from Ada 83!");
end HELLO;


Output:  Hello from Ada 83!
```

Stage 1 — Ada 83 Compiler Front-End

Completed: May 16, 2026 | Implementation Language: Ada 83

All five compiler modules written entirely in Ada 83, compiled by Ada/Ed on the Stage 0 bootstrap platform, and tested successfully with 0 errors. This is the first stage in which the project produces Ada-written compiler code. The Abstract Syntax Tree (AST) is the internal data structure used by the parser, semantic analyzer, and code generator to represent the parsed program. The total implementation is 3,580 lines of Ada 83.

Compiler Modules

Module	Lines	Function
`LEXER.ADA`	720	Lexical analyzer — tokenizes all Ada 83 source
`PARSER.ADA`	1,360	Recursive descent parser — builds full Abstract Syntax Tree (AST)
`SYMTAB.ADA`	380	Scoped hash-table symbol table
`SEMAN.ADA`	660	Semantic analyzer — type checking and name resolution
`CODEGEN.ADA`	460	Code generator — produces formatted Ada 83 output
TOTAL	**3,580**	**Lines of Ada 83 compiler source code**

The Complete Pipeline

```
Source.ADA → LEXER → PARSER → SYMTAB + SEMAN → CODEGEN → Output.ADA
```

Input

```
with TEXT_IO; use TEXT_IO; procedure HELLO is begin PUT_LINE("Hello from Ada 83!"); end HELLO;
```

Generated Output — correctly formatted Ada 83

```
with TEXT_IO;
use TEXT_IO;
procedure HELLO is
begin
   PUT_LINE ("Hello from Ada 83!");
end HELLO;
```

Test Results

Metric	Result
Parse errors	0
Semantic errors	0
Generated output	Valid, correctly formatted Ada 83
Symbols resolved	All names correctly resolved via symbol table

Key Technical Achievements

- Lexer correctly recognizes all 63 Ada 83 reserved words, all operators, identifiers, and all literal types including based numeric literals (e.g. 16#FF#)
- Parser implements full recursive descent covering procedures, functions, packages, all statement and expression types with correct operator precedence
- Symbol table implements scoped hash-table lookup with 17 predefined Ada 83 symbols, correct shadowing across nested scopes, and automatic scope cleanup on close
- Semantic analyzer resolves all name references, checks type compatibility, verifies declaration before use, and reports errors with line numbers
- Code generator produces correctly indented, properly formatted Ada 83 source from the AST, closing the source-to-source pipeline
- All five modules compiled by Ada/Ed with 0 errors and pass unit tests on real Ada 83 programs

Technical Specification

Item	Detail
Host platform	64-bit Windows 10/11
Bootstrap compiler	NYU Ada/Ed 1.11.2 (TDM-GCC 10.3.0 build)
Bootstrap language	C (Ada/Ed source)
Stage 1 language	Ada 83 (ANSI/MIL-STD-1815A)
Ada/Ed validation	ACVC 1.7
Target architecture	Intel 80486 32-bit
Ada/Ed source repo	github.com/softwarepreservationgroup/adaed
Stage 0 Ada source	predef.ada (standard library only)
Stage 1 Ada source	3,580 lines across 5 modules

Next Steps — Stages 2 through 5

Stage 2 will extend the compiler to generate executable Ada machine code rather than source output, then use it to compile its own Ada 83 source code — achieving self-hosting. Once the compiler can compile itself, Ada 95 feature extensions begin in Stage 3, followed by Ada 2022 (ISO/IEC 8652:2023) extensions in Stage 4. The ultimate goal is a compiler written entirely in Ada 2022 that can compile itself and pass the Ada Compiler Acceptance Test Suite (ACATS) validation in Stage 5.

Public Domain Declaration