Bootstrapping

Bootstrapping is used to create compilers and to move them from one machine to another by modifying the back end.

Definition: Bootstrapping in compiler design refers to the process of developing a compiler using an existing compiler for the same or a different programming language.

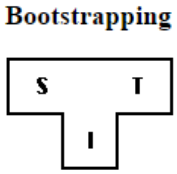
Self-Hosting: The goal of bootstrapping is to create a self-hosting compiler, which is capable of compiling its own source code. Initially, an initial version of the compiler is created using a different language or an existing compiler.

Development Iterations: With the initial version, the compiler's source code is compiled using itself. The resulting compiled version becomes the new compiler, and subsequent iterations use this self-compiled version to improve and refine the compiler.

Advantages: Bootstrapping ensures that the compiler is self-sufficient and can independently compile its own code. It allows for easier maintenance, further development, and evolution of the compiler over time.

A compiler is characterized by three languages

1. Source Language *(S)*
2. Target Language *(T)*
3. Implementation Language *(I)*



Porting

Porting the compiler to the new host computer now only requires that the back end of the source code be rewritten to generate code for the new machine. This is then compiled using the old compiler to produce a cross compiler, and the compiler is again recompiled by the cross compiler to produce a working version for the new machine.

Definition: Porting in compiler design refers to the process of adapting a compiler to run on a different hardware platform or operating system.

Cross-Compilation: In porting, a compiler that is originally designed for a specific hardware platform or operating system is modified to support compilation for a different platform or OS.

Changes and Adaptations: Porting involves making necessary changes and adaptations to the compiler codebase to handle differences in hardware architectures, system libraries, and underlying system APIs.

Testing and Validation: After porting, extensive testing and validation are performed to ensure that the ported compiler functions correctly and produces executable code compatible with the target platform or operating system.

Platform-specific Optimizations: During the porting process, optimizations can be applied to make the compiler more efficient and take advantage of specific features or capabilities of

the target platform.

Cross-Platform Development: Porting enables software developers to write code using a particular language and compiler and deploy it on various platforms without having to rewrite the code from scratch.

Related Posts:

1. Introduction to Compiler
2. Analysis and synthesis model of compilation
3. Lexical Analyzer: Input Buffering
4. Storage Allocation Strategies
5. Type Checking
6. Specification & Recognition of Tokens
7. Front end and back end of the compiler
8. LEX
9. Analysis synthesis model of compilation
10. Data structure in CD
11. Register allocation and assignment
12. Loops in flow graphs
13. Dead code elimination
14. Syntax analysis CFGs
15. L-attribute definition
16. Operator precedence parsing
17. Analysis of syntax directed definition
18. Recursive descent parser
19. Function and operator overloading

20. Storage allocation strategies
21. Equivalence of expression in type checking
22. Storage organization
23. Parameter passing
24. Run time environment
25. Type checking
26. Code generation issue in design of code generator
27. Boolean expression
28. Declaration and assignment in intermediate code generation
29. Code optimization
30. Sources of optimization of basic blocks
31. Loop optimization
32. Global data flow analysis
33. Data flow analysis of structure flow graph (SFG)