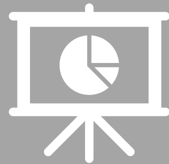
Planejamento da construção de um compilador e a seleção de ferramentas

Prof. Me. Hélio Esperidião

Review 1.1

A fase da análise

está diretamente associada à verificação de se o programa foi escrito corretamente, isto é, de acordo com as regras da linguagem.



A análise está subdividida em 3 etapas

análise Léxica

- Verifica se os nomes das entidades estão corretos.
- Hélio
- Helio
- Elio
- Élio?

Sintática

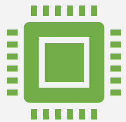
Analisa-se se os comandos estão corretos.

Chama a verificação da frase.

Aqui, não basta escrever as palavras corretamente, importando também a ordem em que elas aparecem

Semântica

nesse ponto, verifica-se o contexto.

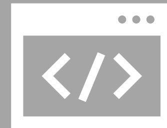


No caso das linguagens de programação, o compilador deverá analisar se os valores envolvidos nos comandos estão compatíveis.

Int x= “helio”;

ferramentas

nesta seção vamos conhecer quais são as ferramentas de apoio existentes no mercado, tais como JFLEX, CUP, JAVACC, Yacc & Lex

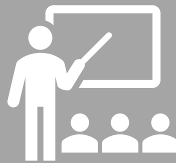


Também iremos analisar as opções de plataforma, tanto para desenvolvimento do compilador como para o código alvo a ser gerado

Essas competências permitirão a você planejar o desenvolvimento dos compiladores.

BNF

É uma notação e foi definida por Peter Naur e melhorada por John Bakus, e por isso o nome Backus-Naur Form (BNF).



Consiste em uma forma matemática de descrever uma linguagem, um modo de definir a gramática da linguagem sem ambiguidade ou divergência.

Primeiramente definimos um símbolo inicial para a gramática, por convenção usamos S (start) e apos esse inicio ditamos regras para substituição desse símbolo por outro. Essas regras são chamadas produções, como por exemplo:

EBNF

É uma extenção a BNF (Extended-BNF) e foi desenvolvida para ampliar a facilidade de leitura e concisão entre as produções.

O símbolo + define uma sequencia de um ou mais elementos da classe marcada.

O símbolo * define uma sequencia de zero ou mais elementos da classe marcada.

Chaves “{” “}” podem ser usadas para agrupar elementos e colchetes “[” “]” podem ser usados para indicar elementos opcionais.

É importante saber que toda linguagem definida em uma EBNF pode ser transformada para as regras da BNF sem perder a suas características e respeitando a gramática.

EBNF

é uma metalinguagem, ou seja, uma linguagem para criar outra linguagem então, por princípio, é uma linguagem

Assim, os cientistas da computação construíram aplicativos que interpretam a especificação EBNF de uma linguagem e automaticamente criam algumas classes (objetos) para reconhecer se a entrada é válida

etapas do projeto do compilador

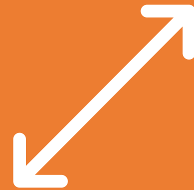
- Esses aplicativos agilizam e sistematizam algumas etapas do projeto do compilador, assim o desenvolvedor pode focar o seu trabalho na construção da gramática e na integração das fases do projeto.

começar do zero

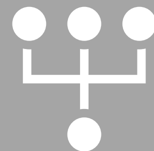
não é preciso começar do zero para criar compiladores nos dias atuais

diferentemente do que acontecia nas décadas de 50 e 60

o primeiro compilador



Ele foi escrito em código de máquina e era muito trabalhoso desenvolver, mas tinha como vantagem a rapidez e serviu como base para todos os outros, por meio do processo de bootstraping



Bootstrapping é o processo de começar uma empresa do zero

- Os compiladores construídos com esta técnica são conhecidos como autocompiláveis, pois em alguma fase da sua construção utilizam a mesma linguagem de programação na qual foram implementados.

autocompiláveis

Vamos entender como funciona o processo de bootstraping, em 3 passos

1º passo: para a linguagem ‘A’, escrevemos o compilador na linguagem de máquina ‘M’

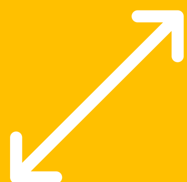
2º passo: agora, para a linguagem ‘A’, escrevemos um outro compilador na própria linguagem ‘A’

3º passo: a saída do 2º passo será entrada para o primeiro compilador, aquele construído no 1º passo

Passo a passo

Seguindo esses passos, tem-se o código de máquina a partir de um compilador feito na própria linguagem.

Pelo processo de bootstrapping, a partir de dois compiladores é possível escrever um compilador em qualquer linguagem de programação.



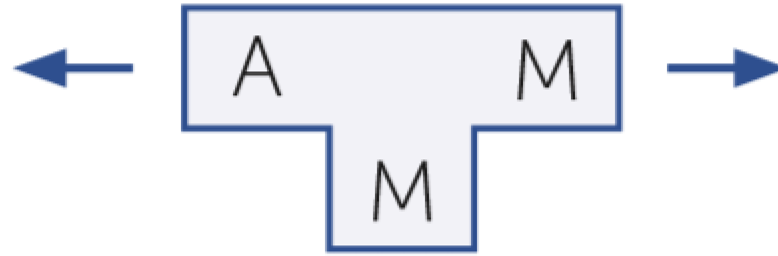
O importante a se considerar que quanto mais passos forem necessários para alcançar o código alvo, mais demorada será a compilação

Bootstrapping

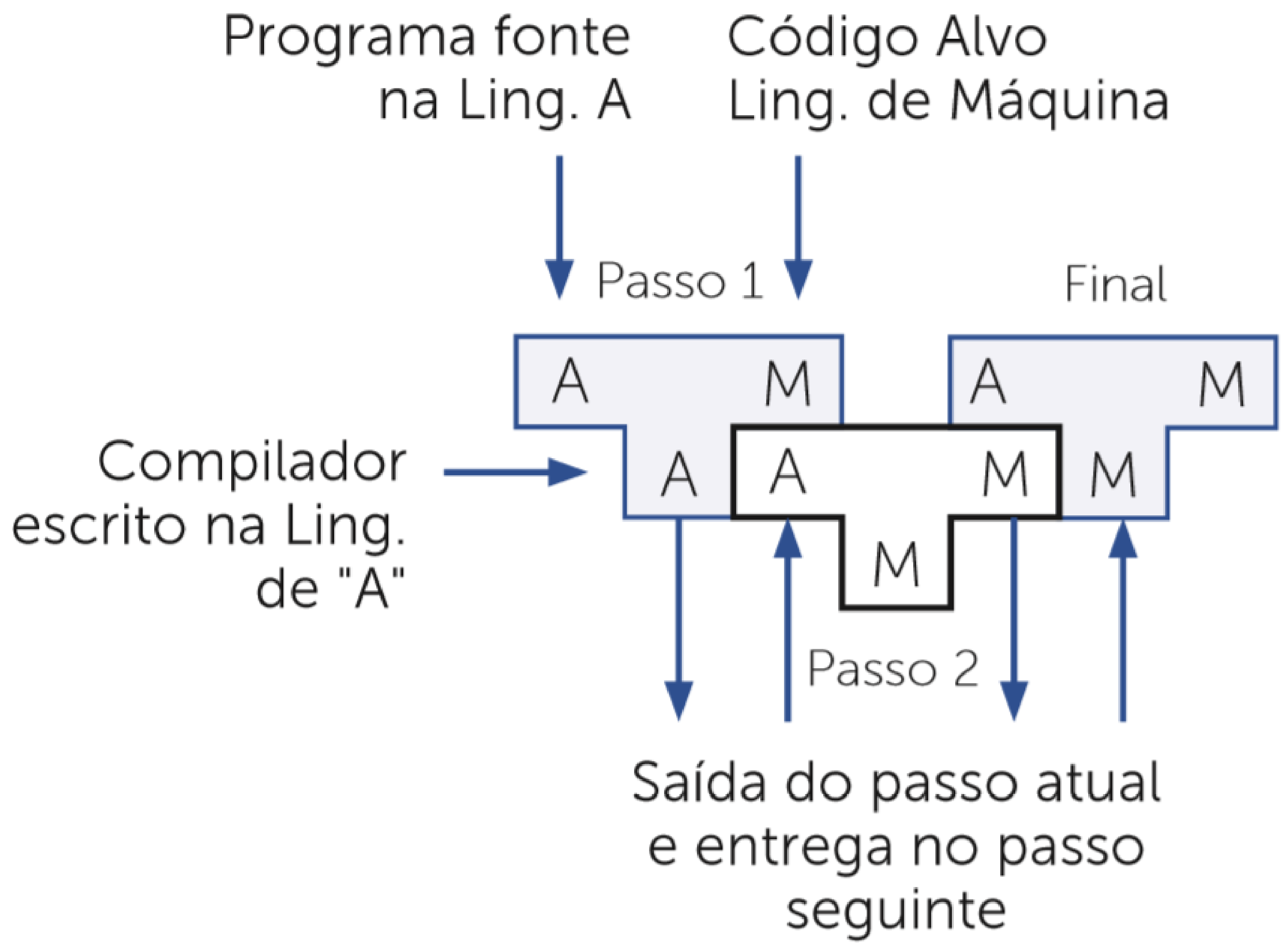
Processo utilizado pelos compiladores autocompiláveis, isto é, em algum momento é utilizada a própria linguagem de programação na qual o compilador foi implementado.

Vamos ver um exemplo com o uso de um diagrama em T.

O primeiro compilador para a linguagem 'A' foi escrito em código de máquina


Programa fonte
na Ling. A
A
M
M
Código Alvo
Ling. de Máquina
Compilador
escrito na Ling.
de Máquina

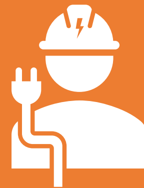
Exemplo de bootstrapping



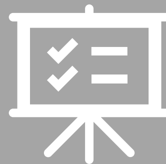
cross-compilers

- Além dos compiladores autocompiláveis, temos os que são cross-compilers (compiladores cruzados), que usam a técnica de bootstrapping e são escritos em um ambiente, mas rodam em outro

Ferramentas



Basicamente, essas ferramentas estão divididas em: geradores de analisadores léxicos, geradores de analisadores sintáticos e geradores de código.



Os geradores de analisadores léxicos permitem a automatização do processo de criação de autômatos e o reconhecimento das sentenças regulares a partir da especificação na notação EBNF.

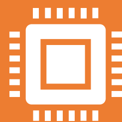
termos que precisam estar bem claros

Token: é o nome da produção da gramática

Lexema: é o elemento do token. Se comparássemos a um programa, poderíamos dizer que token é o nome da variável e o lexema o conteúdo da variável.

Scanner: é o gerador de analisador léxico, por exemplo: LEX, JFLEX. O scanner lê a especificação em um padrão EBNF e gera um programa que analisa um arquivo fonte (programa) escrito de acordo com a especificação, por exemplo: o Lex gera em C e o JFLEX gera em Java

termos que precisam estar bem claros



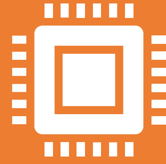
Lexer: é o analisador léxico. O programa gerado pelo scanner.

Parser: é o gerador de analisador sintático, por exemplo: Yacc, CUP.

O parser lê a especificação da GLC (gramática livre de contexto) no padrão EBNF, recebe os tokens analisados pelo lexer e gera um programa que analisa a sintaxe de um arquivo fonte (programa) escrito de acordo com a especificação GLC

por exemplo: o Yacc gera em C e o CUP gera em Java.

termos que precisam estar bem claros



Parsing: é o analisador sintático. O programa gerado pelo parser

Assembly: linguagem de baixo nível.

Assembler: é o compilador para a linguagem assembly. Faz a passagem do código fonte para o código alvo em uma passada, também chamado de montador.

Analisadores léxico

O primeiro e mais conhecido gerador de analisadores léxicos é o Lex.

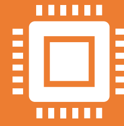
foi projetado por Mike Lesk e Eric Schmidt para trabalhar com o Yacc, um gerador de analisador sintático, e ambas ferramentas rodam na plataforma UNIX e geram código em C

Essas ferramentas trabalham juntas para construírem o analisador sintático (o parser).

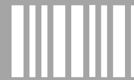
scanners

- Os geradores de analisadores léxicos, também conhecidos como scanners, leem a especificação em um padrão EBNF e geram um programa que analisa o arquivo fonte escrito de acordo com a especificação, por exemplo: o Lex gera em C e o JFLEX gera em Java

Flex

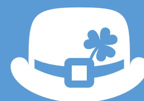


Outro representante dos scanners é o Flex, cujo scanner tem a vantagem de gerar analisadores léxicos mais rápidos que o Lex e acompanha a sintaxe do Lex com pequenas variações.



O código gerado também é em C.

Existem, também, geradores para Java. São o JAVACC e JFLEX. O JAVACC é uma ferramenta completa, com o scanner e o parser.



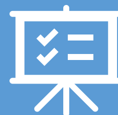
Há, ainda, a dupla JFLEX&CUP. O JFLEX é o scanner e trabalha em conjunto com o CUP que é o parser. A dupla JFLEX & CUP é uma adaptação do Lex & Yacc para o Java, com pequenas variações

Reflita

Quais as vantagens de construirmos um compilador do zero?

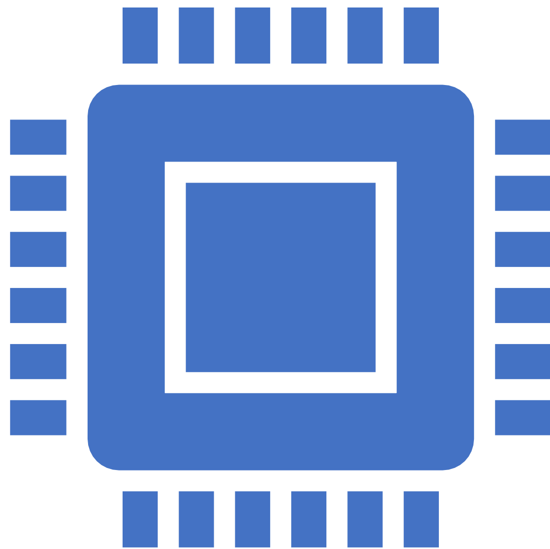
E no caso de usar ferramentas de apoio, haveria vantagens ou desvantagens?

Se você desejasse construir um compilador usando Java, qual ferramenta usaria? Caso a escolha seja utilizar a linguagem Python, qual ferramenta de apoio poderia usar?



Se lembrarmos que a técnica de bootstrapping permite criar compiladores em qualquer linguagem de programação, a escolha será limitada apenas pela plataforma que irá utilizar e pela linguagem utilizada pelas ferramentas de apoio, não é verdade?

Front end



- As ferramentas, os scanners e os parsers compõem o frontend do compilador e estão associados à parte de análise.
- A parte da síntese, em que é gerado o código alvo, constitui o backend.

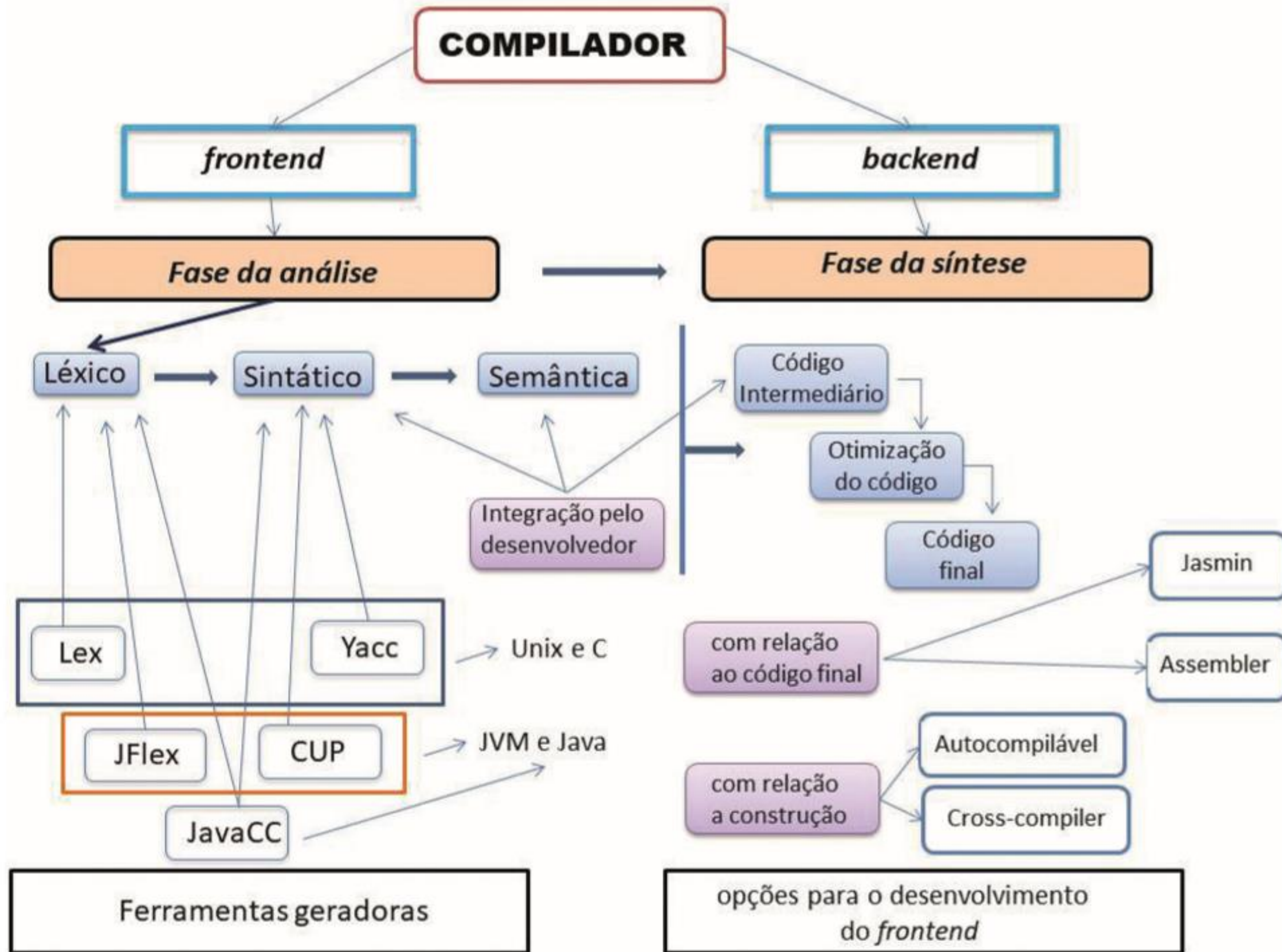
Back End Generator

Podemos utilizar, ainda, simuladores para a geração do código para a máquina final, como é o caso da ENS2001, ou usar TASM e TLink, que são montadores para a linguagem assembly.

Para o Java, temos o Jasmin, que é um assembler (montador) para a Java Virtual Machine (JVM).

- Figura 1.5 nos mostra um mapa com essas opções, permitindo uma visão geral dessas alternativas, apresentando um mapa conceitual do processo de construção do compilador. O frontend é a fase da análise, composta pelo léxico, sintático e semântico. Para a construção da análise léxica e sintática, caso seja utilizada a plataforma UNIX e a linguagem C para construção do compilador, poderão ser usadas as ferramentas LEX & YACC; caso sejam utilizados o ambiente JVM e a linguagem JAVA, pode-se utilizar as ferramentas JFLEX & CUP ou JAVACC. Para os analisadores gerados, quaisquer que tenham sido as ferramentas de apoio utilizadas, elas deverão ser integradas pelo desenvolvedor para conclusão da análise semântica e passagem para a fase de síntese. O backend do compilador consiste na fase de síntese, mas, para ela, não temos ferramentas de apoio. A fase de síntese está subdividida em código intermediário, otimização do código e

- geração do código final. No planejamento do compilador, deve-se decidir se a implementação será autocompilável (uso da técnica de bootstrapping) ou por cross-compillers. Para geração do código de máquina será necessário usar algum montador (Assembler) ou, caso seja um compilador híbrido, usar uma máquina virtual, por exemplo Jasmin para o JVM.


COMPILADOR
frontend
backend
Fase da análise
Fase da síntese
Léxico
Sintático
Semântica
Código Intermediário
Otimização do código
Código final
Integração pelo desenvolvedor
Jasmin
Lex
Yacc
Unix e C
com relação ao código final
Assembler
JFlex
CUP
JVM e Java
Autocompilável
com relação a construção
JavaCC
Cross-compiler
Ferramentas geradoras
opções para o desenvolvimento do frontend

bom compilador

um bom compilador: gera o código alvo correto; faz a análise do código fonte de acordo com a especificação da linguagem;

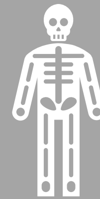
é capaz de lidar com programas de qualquer tamanho e de usar algoritmos adequados para a otimização dos códigos e do gerenciamento da memória; e, finalmente, é facilmente portável para diferentes plataformas.

meta a ser atingida

- Após conhecer as etapas envolvidas no processo de construção do compilador e as qualidades a que ele deve atender, será necessário direcionarmos nossa visão para a meta a ser atingida por ele

perguntas deverão ser respondidas

Nesse ponto do planejamento, algumas perguntas deverão ser respondidas antes de ser iniciada a construção do compilador.



Veja os itens que deverão ser analisados pelo projetista:

Finalidade do compilador a ser criado

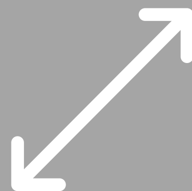
Eficiência

(a) Com relação ao processo de compilação, se necessitarmos rapidez na compilação, será importante analisar o número de passagens que serão feitas.

Vale lembrar que a implementação por interpretação também resulta em baixa eficiência nesses casos.

Eficiência

(b) Com relação à eficiência do código alvo resultante, o cuidado que deverá ser tomado nesse caso está relacionado aos algoritmos de otimização do código e ao gerenciamento da memória.



Apesar de o tamanho dos executáveis não ser mais um grande problema, isso tem grande relevância caso os usuários dos compiladores forem gerar aplicações para softwares embarcados, caso em que o tamanho do código gerado é limitado.

Plataforma

- É preciso considerar para qual plataforma o código alvo irá rodar, qual será a plataforma do desenvolvimento e a portabilidade tanto do código alvo como do compilador.

Plataforma para o código alvo

- (a) Plataforma para o código alvo: Se o compilador e o código alvo usarem a mesma plataforma, devemos fazer a implementação cross-compiler, caso contrário, outras opções podem ser usadas, tal como a autocompilável ou a autoresidente;

Plataforma do desenvolvimento

- dependendo da plataforma do desenvolvimento, teremos opções especificas com relação às ferramentas de apoio, e, se a linguagem do desenvolvimento não tiver ferramenta que agilize sua construção, ele terá que ser feito manualmente e/ou teremos que usar mais passos, com a aplicação da técnica de bootstapping

Portabilidade

- Atualmente, a portabilidade, tanto do código alvo como do próprio compilador, deve ser levada em consideração, afinal, os clientes querem ter liberdade para usar os aplicativos em diversas plataformas e não ter limitações para uso, e o fabricante, por sua vez, não quer limitar seu mercado de venda criando aplicações apenas para uma plataforma.

Linguagem do desenvolvimento

A escolha da linguagem na qual será construído o compilador abrirá ou não a possibilidade do uso de ferramentas de apoio, além de definir o grau de eficiência do processo de compilação.

Tanto o uso de linguagem interpretada como uma compilação de múltiplos passos geram lentidão no processo

Pesquise mais

- Desenvolver um compilador é construir um programa complexo, portanto, estude mais sobre:
  - Linguagem de programação C, C++ e Java. b) Notação EBNF, gramática regular e linguagens livres de contexto. c) Geradores de analisadores léxicos e sintáticos. Isso o motivará e ajudará no seu aproveitamento dos estudos, trazendo um diferencial para a sua formação profissional, já que este é um tema de grande complexidade e com um mercado promissor, com poucos especialistas com visão prática.