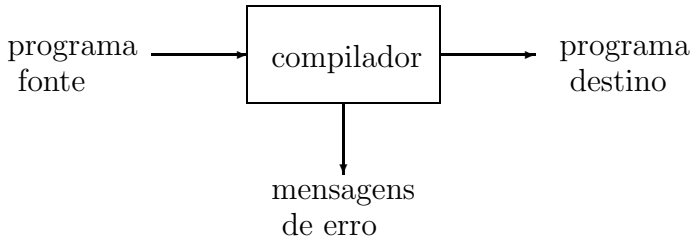
Capítulo 1

Introdução

Um compilador é um programa que lê um programa escrito numa dada linguagem, a linguagem objecto (fonte), e a traduz num programa equivalente numa outra linguagem, a linguagem destino. Como parte importante neste processo o compilador deve dar informação acerca dos eventuais erros no programa objecto (fonte).



Dada a multiplicidade de linguagens fonte diferentes e a multiplicidade de linguagens destino a construção de compiladores para todos esses tipos de linguagens seria uma tarefa dantesca.

> O primeiro compilador de FORTRAN (anos 50) exigiu um esforço de 18 pessoas-ano para ser implementado

Actualmente a utilização de ferramentas computacionais apropriadas assim como de técnicas especializadas a construção de compiladores tornou-se uma tarefa mais fácil de executar.

Existem muitos programas que sem tomarem o nome de compilador usam as técnicas próprias dos compiladores, ou são mesmo compiladores, não no sentido usual das linguagens de programação, mas para outros tipos de linguagens fonte e linguagens objecto. Vejamos sumariamente alguns desses programas.

[1](Versão 1.6)

Formatadores ("Pretty Printers") Os "pretty printers" são programas que recebem um texto (linguagem objecto) de um programa escrito numa dada linguagem de programação, e produzem um outro texto (linguagem destino) formatado de forma a que a estrutura própria dos vários construtores da linguagem de programação é realçada.

Verificadores de Programas ("Static Checkers") Os "static checkers" analisam o texto e produzem mensagens de erro/aviso sem chegarem a produzir um texto final, isto é implementam somente a fase de análise e tratamento de erros dos compiladores.

Um exemplo de um programa deste tipo é o verificadores de texto em LaTeX designado por *Lacheck*, este programa analisa textos LaTeX produzindo mensagens de erro e de aviso apropriadas.

Um outro exemplo é nos dado pelo programa *lint*, o verificador de programas para o sistema *GNU C*.

Interpretadores os interpretadores são programas que recebem como entrada um texto (expressão ou programa) escrito numa dada linguagem de programação e produzindo o efeito desejado sem que para isso produzam um programa em linguagem destino.

Formatadores de Texto os formatadores de texto são programa que recebem textos (documentos) contendo comandos de formatação produzindo como resultado um texto (documento) formatado pronto para ser impresso/visualizado.

Exemplos de programas deste tipo são os formatadores de texto LaTeX, *nroff*, e *troff*.

1.1 O Contexto de um Compilador

Aquilo que é usualmente designado por um compilador é de facto um conjunto de programas que no seu conjunto formam o compilador, isto é designa-se pelo mesmo nome o conjunto e um dos seus elementos. É de notar que nem sempre os programas que a seguir se vão descrever estão presentes num sistema de compilação (compilador), e que, por outro lado, na maior parte dos casos a separação do sistema de compilação nas suas várias componentes não é tornada explícita ao utilizador.

pré-processador processa directivas; retira os comentários; agrupa ficheiros se tal for necessário, entre outras tarefas. No sistema *GNU C* temos o *cpp*, "the GNU C-Compatible Compiler Preprocessor".

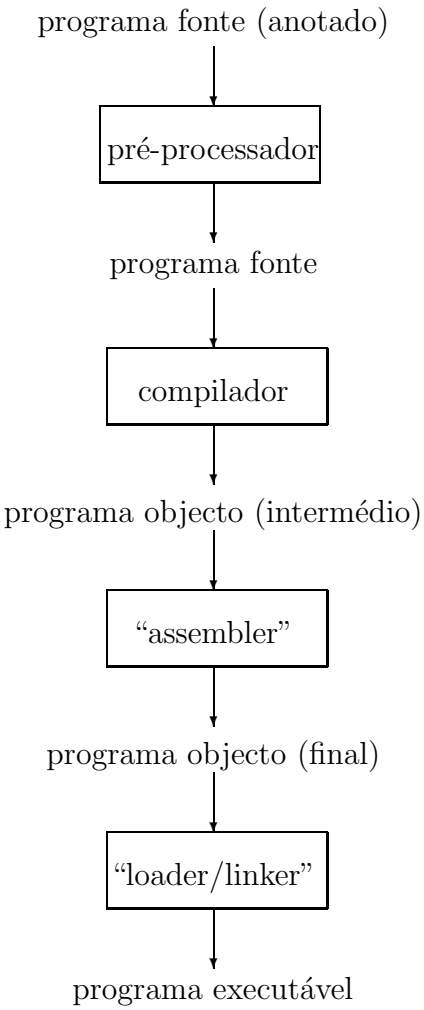
compilador faz a análise do texto escrito na linguagem fonte e faz a sua transcrição para a linguagem destino, por razões de economia (possibilidade de escrever um compilador que seja adaptável a diferentes

sistemas computacionais destino) a linguagem destino é tão somente uma linguagem (genérica) intermediária. No sistema *GNU C* temos o *gcc*, "the GNU project C Compiler".

"assembler" faz a transcrição da linguagem intermédia para a linguagem final (máquina), e um programa que está fortemente ligado a um (e um só) sistema computacional.

"loader/linker" no caso de que se querer um programa executável os "loader/linker" fazem a junção do código máquina produzido pelas anteriores fases a um conjunto de serviços ("run-time routines") que permitem a criação de um programa independente. No sistema *GNU C* temos o *ld*, "the GNU linker".

Graficamente:



1.2 Fases de um Compilador

Um compilador pode-se dividir em duas fases:

- uma fase de análise na qual o texto do programa fonte é separado em todas as suas partes constituintes, e na qual se cria uma representação intermédia do programa fonte.
- uma fase de síntese na qual se constroí o programa destino a partir da representação intermédia.

1.2.1 Análise

As fases de análise lexical, análise sintáctica, e análise semântica, formam a componente de análise de um compilador, também designado por "front end".

Análise Lexical leitura da sequência de caracteres (ficheiro) que constituem o programa fonte, convertendo-a numa sequência de palavras.

Análise Sintáctica recebe a sequência de palavras da fase anterior e converte-a numa sequência de frases.

Análise Semântica recebe a sequência de frases da fase anterior e verifica a sua correcção semântica.

Vejamos um exemplo, seja a seguinte instrução de atribuição:

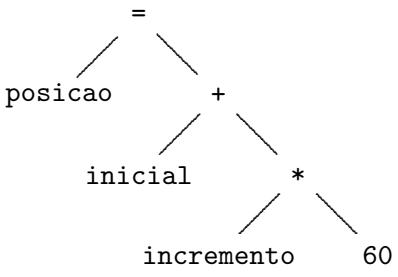
```
posicao = inicial + incremento * 60 <eof>
```

Na fase de análise léxica esta sequência de caracteres é separada numa lista de palavras:

```
(posicao,=,inicial,+,incremento,*,60)
```

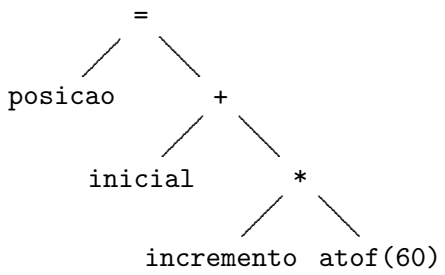
os espaços foram ignorados.

De seguida temos a fase de análise sintáctica em que se tenta construir uma frase correcta com a lista de palavras produzidas pela fase anterior. É usual construir-se uma estrutura em árvore para representar a frase obtida.



A forma como a árvore é construída é determinada por um conjunto de regras (diagramas de sintaxe do *C*, ...).

Finalmente tem-se a fase de análise semântica em que, no caso presente se vai tentar verificar a validade da frase no que diz respeito aos tipos das entidades utilizadas. Por exemplo, se um dos identificadores presentes na expressão é do tipo `real`, então é necessário converter `60` para a sua representação `real`. Obtinha-se a árvore seguinte:



1.2.2 Tabela de Símbolos

Uma das consequências do que se acabou de dizer é que a informação acerca das palavras que o programa contêm tem de fluir entre as várias fases do compilador.

A gestão da *Tabela de Símbolos* é pois de primordial importância, e vai cruzar todas as fases do compilador. A tabela de símbolos vai conter uma entrada para cada uma das palavras que foram identificadas pelo analisador léxico contendo além da palavra propriamente dita todos os atributos que lhe são próprios.

1.2.3 Gestão de Erros

Outra tarefa que é comum a todas as fases do compilador é a da *Gestão dos Erros*. Trata-se de detectar os erros enviando uma mensagem apropriada para o utilizador: local do erro; tipo do erro; causa provável.

Além disso, e para aumentar a produtividade, é importante tentar recuperar do erro automaticamente de forma a poder continuar a tarefa de compilação até ao máximo possível. Por exemplo, ao detectar a falta de um "`;`" entre duas instruções correctas o compilador deverá fazer a introdução automática da palavra em falta de forma a poder prosseguir com a compilação.

1.2.4 Síntese

As fases seguintes têm a ver com a geração do código, a fase de síntese é também designada por "back end".

A fase de geração do código intermédio tem como objectivo a transformação da frase (árvore) construída na fase de análise numa frase numa linguagem (código) intermédio. A finalidade da geração de um código intermédio em vez do código final tem a ver com a necessidade de tornar mais fácil a construção de compiladores para diferentes sistemas finais, até ao código intermédio o compilador é um programa que é independente de um dado sistema computacional, só na fase de geração do código final é que se vai passar a trabalhar ao nível de máquina alvo produzindo código para um só tipo de máquina.

As fases seguintes são a *optimização do código* na qual se pretende aplicar várias estratégias de optimização ao código anteriormente produzido, e a fase de *geração do código máquina*, na qual se faz a tradução do código intermédio para o código máquina do sistema alvo.

Graficamente temos: