1. INTRODUÇÃO

TRADUTORES E COMPILADORES

Um *tradutor* é um programa que recebe como dado de entrada um programa escrito em uma linguagem de programação (a *linguagem fonte*) e produz como saída de seu processamento um programa escrito em outra linguagem (a *linguagem objeto*).

Se a linguagem fonte é uma linguagem de alto nível como Fortran, Pascal ou C, e a linguagem objeto é uma linguagem de baixo nível como a linguagem de montagem ("assembly") ou de máquina, o tradutor é chamado de *compilador*.

Por esse enfoque, a execução de um programa escrito em linguagem de programação de alto nível é basicamente um processo de dois passos, como mostrado na figura 1.1. O programa fonte deve primeiro ser *compilado*, isto é, traduzido para a linguagem objeto, para, em seguida, ser carregado na memória e executado.

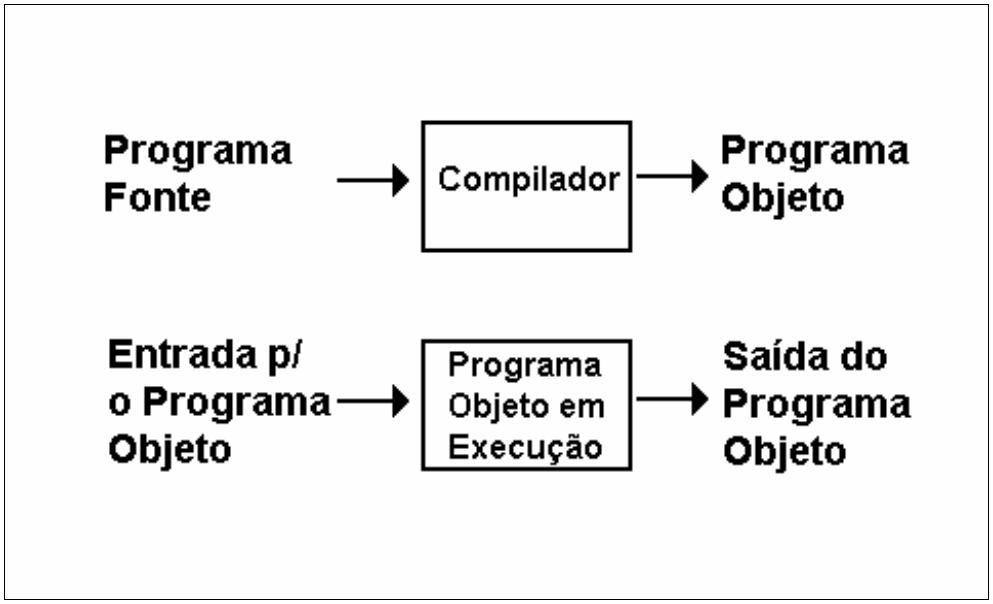

Figura 1.1 - Compilação e Execução de um programa

OUTROS TRADUTORES

Interpretadores

Certos tipos de tradutores transformam uma linguagem de programação (LP) em uma linguagem simplificada, chamada de *código intermediário*, que pode ser diretamente "executado" por um programa chamado *interpretador*. Nós podemos imaginar o código intermediário como uma linguagem de máquina de um computador abstrato projetado para executar o código fonte. Por exemplo, Basic, Prolog e Java, são freqüentemente interpretadas.

Interpretadores são, em geral, menores que compiladores e facilitam a implementação de construções complexas em LPs. Entretanto, o tempo de execução de um programa interpretado é geralmente maior que o tempo de execução desse mesmo programa compilado.

Montadores ("Assemblers")

***Montadores* traduzem programas escritos em linguagem de montagem nos correspondentes programas escritos linguagem de máquina (0s e 1s).**

Pré-processadores

***Pré-processadores* traduzem programas escritos em linguagens de alto nível em outros programas escritos também em linguagens de alto nível (p.ex. o pré-processador da linguagem C).**

Macroprocessadores

***Macroprocessadores*, semelhantes aos pré-processadores, traduzem programas escritos em linguagens de alto nível em outros programas também escritos em linguagens de alto nível, tendo também a capacidade de processamento de macro-instruções. Uma *macro-instrução* é um nome simbólico associado à um conjunto de instruções de uma LP que pode ser usado como referência a esse conjunto de instruções dentro do programa. Uma macro-instrução pode ou não ter parâmetros. Veja um exemplo de uso na figura 1.2.**

```
#define MIN( A, B )     ( (A) <= (B) ? (A) : (B) )
. . .
    int     x, y, z;

    . . .

    z = MIN( x, y );
```

Figura 1.2 - Exemplo de uso de macro-instrução

ESTRUTURA DE UM COMPILADOR

Como dissemos anteriormente, um compilador recebe como entrada um programa fonte e produz como saída um programa objeto na forma de um conjunto de instruções em linguagem de máquina (ou, mais comum hoje em dia, em linguagem de montagem).

Esse processo é tão complexo que não é razoável, do ponto de vista lógico e de implementação, considerá-lo como sendo desenvolvido em um único passo. Por essa razão, é costume partir o processo de compilação em uma série de subprocessos chamados fases, como podemos ver na figura 1.3.

Uma *fase* é uma operação que toma como entrada uma representação do programa fonte e produz como saída uma outra representação.

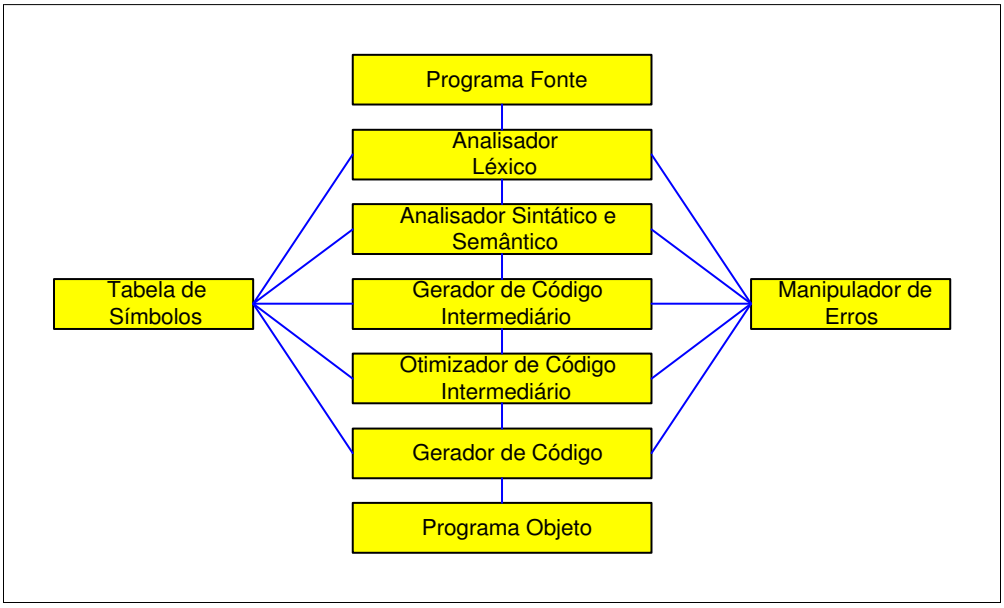

Figura 1.3 - Estrutura de um Compilador

Na primeira fase, um módulo chamado analisador léxico (*scanner*), lê o programa fonte caracter a caracter, agrupando a seqüência de caracteres lidos em grupos de símbolos (*tokens*). Os símbolos são as palavras-chaves tais como BEGIN e THEN, nomes (identificadores) de variáveis e procedimentos tais como X e SOMA, operadores tais como + e <=, e símbolos de pontuação tais como parênteses e ponto-e-vírgula.

A saída do analisador léxico é uma seqüência de símbolos que é passada para a próxima fase, o analisador sintático. Os símbolos nessa seqüência são geralmente representados por códigos (valores inteiros).

O analisador sintático (*parser*) agrupa os símbolos recebidos do analisador léxico em estruturas sintáticas. Os três símbolos representando A + B poderiam ser agrupados em uma estrutura sintática chamada *expressão*. Expressões poderiam ser posteriormente agrupadas para formar comandos, e assim por diante.

O analisador semântico verifica se estruturas sintáticas pelo analisador sintático, embora corretas sintaticamente, têm significado admissível na linguagem. Por exemplo, A + B pode ser uma expressão sintaticamente correta, mas pode não ter significado em muitas linguagens se A for inteiro e B for caracter.

O gerador de código intermediário usa as estruturas produzidas pelo analisador sintático e verificadas pelo analisador semântico para criar uma seqüência de instruções simples dita código intermediário (está entre a linguagem de alto nível e a linguagem de baixo nível).

O otimizador de código (independente de máquina) é um módulo opcional (presente na grande maioria dos compiladores) que objetiva melhorar o código intermediário de modo que o programa objeto produzido ao fim da compilação seja menor (ocupe menos espaço de memória) e/ou mais rápido (tenha tempo de execução menor). A saída do otimizador de código é um novo código intermediário.

O gerador de código produz o código objeto final, tomando decisões com relação à alocação de espaço para os dados do programa, selecionando a forma de acessá-los, definindo que registradores da UCP serão usados, etc. Projetar um gerador de código que produza programas objeto verdadeiramente eficientes é uma das tarefas mais difíceis no projeto de um compilador.

A maioria dos compiladores atuais tem, como parte integrante de seu gerador de código, um módulo adicional de otimização de código dependente de máquina que tem por objetivo melhorar o código de máquina produzido para melhor aproveitar os recursos específicos da arquitetura para a qual foi gerado.

O módulo de gerência de tabela de símbolos tem por função guardar informações a respeito de todos os nomes usados pelo programa e registrar informações importantes associadas a cada um, tais como seu tipo (inteiro, real, etc.), tamanho, escopo, etc. A estrutura de dados usada para registrar essas informações é chamada tabela de símbolos.

Finalmente, o manipulador de erros é ativado sempre que for detectado um erro no programa fonte. Ele deve avisar o programador da ocorrência do erro emitindo uma mensagem, e ajustar-se novamente à informação sendo passada de fase a fase de modo a poder completar o processo de compilação (mesmo que não seja mais possível gerar código objeto, a análise léxica e sintática deve prosseguir até o fim).

Os módulos de gerência de tabela de símbolos e manipulação de erros interagem sempre com todos os outros módulos do compilador.

Na construção de um compilador, uma ou mais fases são combinadas em um único bloco chamado *passo*. Um passo lê o programa fonte ou a saída gerada pelo passo anterior, fazendo as transformações específicas de suas fases, gravando sua saída em um arquivo temporário que vai ser lido pelo passo posterior.

Quando várias fases são agrupadas em um único passo, suas operações podem ser intercaladas com o devido controle de alternância entre elas.

O número de passos e o agrupamento de fases em cada passo são geralmente definidos em função da estrutura da LP e das características da arquitetura para a qual o compilador vai gerar código.