Matheus Goulart de Freitas **RA: 16272126**

Dericson Pablo Calari Nunes **RA: 16399426**

Compiladores : Front-end e Back-end

O front-end é composto por três componentes:

- Analisador léxico;
- Analisador sintático;
- Analisador semântico;

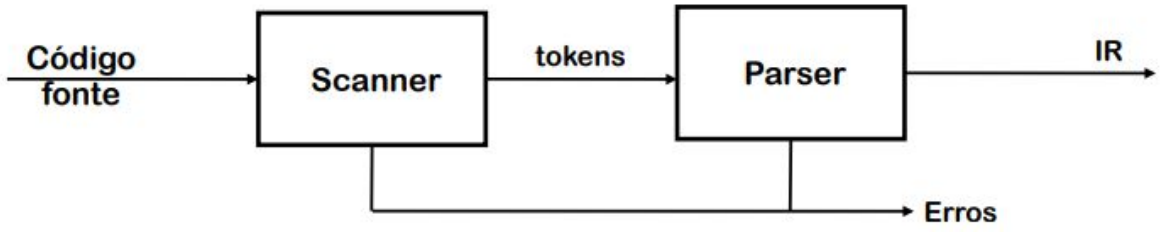
O back-end também executa três tarefas básicas:

- Seleção de instruções;
- Escalonamento de instruções;
- Alocação de registradores;

1) Descreva cada um destes elementos em até 10 linhas (arial, 11).
2) Qual o papel da RI (representação intermediária) durante a compilação?

Respostas:

1) FRONT-END



a) Analisador léxico: É um programa que implementa um autômato finito, reconhecendo strings como símbolos válidos de uma linguagem. A implementação de um analisador léxico requer uma descrição do autômato que reconhece as sentenças da gramática ou expressão regular de interesse. É a primeira etapa do processo de compilação e seu objetivo é dividir o código fonte em símbolos, preparado-o para a Análise Sintática. Neste processo pode-se destacar três atividades como fundamentais: Extração e classificação dos tokens; Eliminação de delimitadores e comentários; Recuperação de Erros.

b) Analisador sintático:

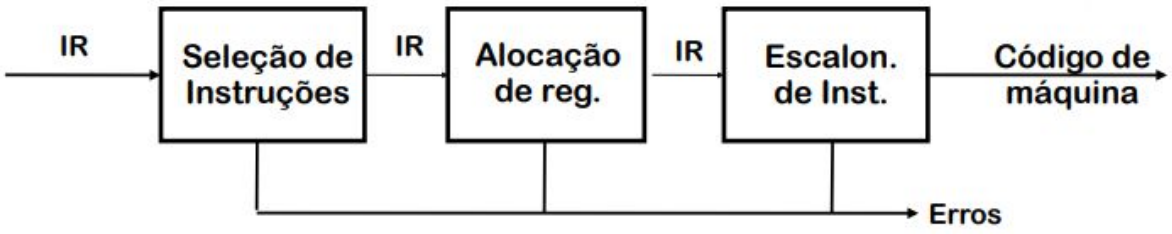
Tem como tarefa principal determinar se o programa de entrada representado pelo fluxo de tokens possui as sentenças válidas para a linguagem de programação.

A análise sintática é a segunda etapa do processo de compilação e na maioria dos casos utiliza gramáticas livres de contexto para especificar a sintaxe de uma linguagem de programação.

c) Analisador semântico: A análise semântica é responsável por verificar aspectos relacionados ao significado das instruções, essa é a terceira etapa do processo de compilação e nesse momento ocorre a validação de uma série regras que não podem ser verificadas nas etapas anteriores.

É importante ressaltar que muitos dos erros semânticos tem origem de regras dependentes da linguagem de programação. As validações que não podem ser executadas pelas etapas anteriores devem ser executadas durante a análise semântica a fim de garantir que o programa fonte esteja coerente e o mesmo possa ser convertido para linguagem de máquina. A análise semântica percorre a árvore sintática relaciona os identificadores com seus dependentes de acordo com a estrutura hierárquica.

BACK-END



d) Seleção de instruções:

A seleção de instruções produz código rápido e compacto e também aproveita recursos da máquina como modos de endereçamento, normalmente visto como um problema de casamento de padrões e métodos ad hoc, casamento de padrões, programação dinâmica, forma da IR influencia escolha da técnica.

Perdeu importância com arquiteturas modernas, Processadores eram mais complicados, ortogonalidade dos processadores RISC simplificou esse problema, desta forma os compiladores usam soluções aproximadas.

e) Escalonamento de instruções:

Refere-se a atribuição de operações e recursos físicos e ordenação específica dos mesmos. Pode ser aplicado a uma linguagem intermediária tanto antes da alocação de registradores (prepass) quanto depois (postpass). No prepass se explora o paralelismo e aumenta a possibilidade de spill code enquanto no postpass é restringido a exploração do paralelismo e não aumenta a ocorrência de spill code.

f) Alocação de registradores:

Mapeia os registradores virtuais para os físicos da arquitetura alvo, gerencia um conjunto limitado de recursos, pode mudar a escolha de instrução e inserir LOADs e STOREs (o que afeta a seleção e escalonamento). Uma alocação ótima é NP-completa na maioria dos casos.

2) Representação intermediária: Tomando um compilador como um modelo de análise-e-síntese, podemos considerar que seus módulos iniciais ("front-end modules") traduzem um programa fonte em uma representação (linguagem) intermediária, a partir da qual seus módulos finais ("back-end modules") geram o código objeto final. A geração de um código intermediário requer um passo a mais para a tradução, tornando o processo um pouco mais lento. Embora um programa fonte possa ser traduzido diretamente para a linguagem objeto, o uso de uma representação intermediária, independente de máquina, tem as seguintes vantagens:

1. Reaproveitamento de código, facilitando o transporte de um compilador para diversas plataformas de hardware; somente os módulos finais precisam ser refeitos a cada transporte.
2. Um otimizador de código independente de máquina pode ser usado no código intermediário.

Pode-se pensar nessa representação intermediária como um programa para uma máquina abstrata. A representação intermediária deve possuir duas propriedades importantes: ser fácil de produzir e fácil de traduzir no programa alvo. A diferença básica entre o código intermediário e o código objeto final é que não são especificados detalhes da máquina alvo, tais como quais registradores serão usados, quais endereços de memória serão referenciados.