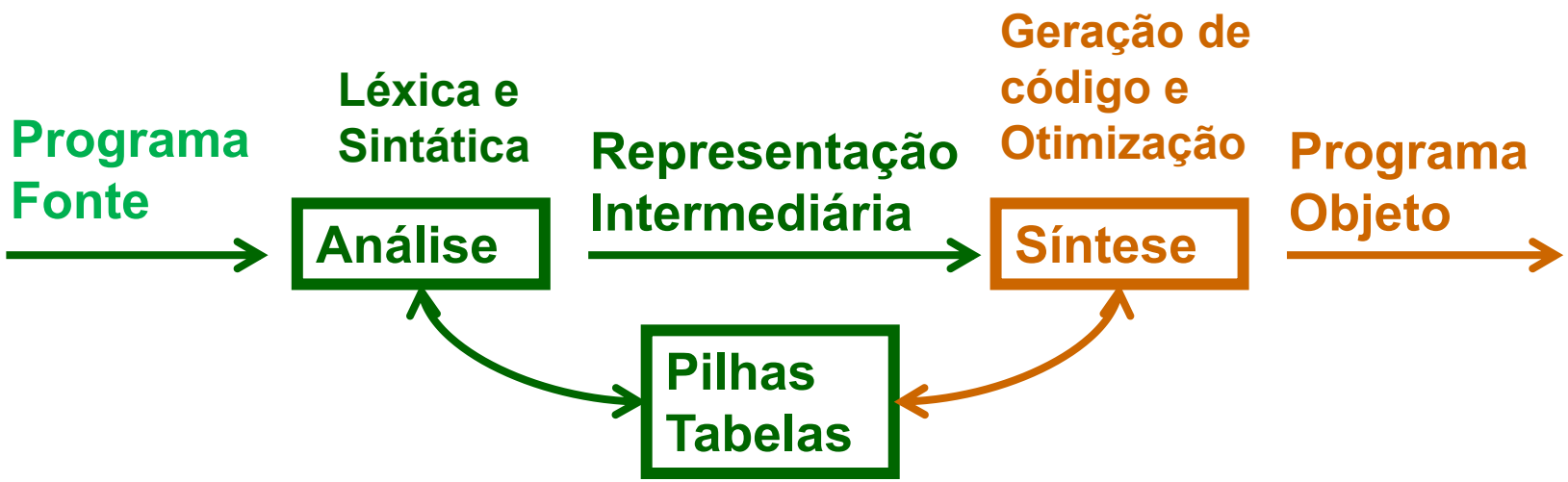
A estrutura global do compilador:



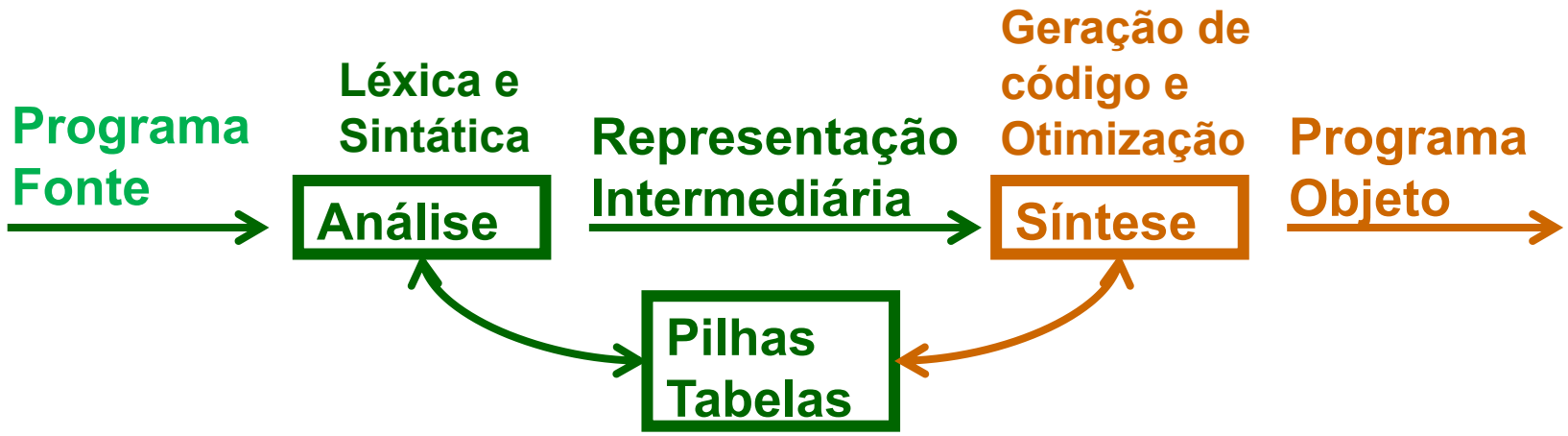
Palavra Compilador, surgida nos anos 50, entendida como:

- Inicialmente, um **processo de composição de várias rotinas** de uma biblioteca para **gerar um programa**;

- 60 anos atrás, este **processo de tradução** era chamado de **programação automática**.

- hoje, considerado o **processo de tradução** de um **programa em linguagem fonte** para um **programa em linguagem objeto**.

A estrutura global do compilador:



Processo de tradução: 2 tarefas essenciais: Análise, Síntese

Análise: o texto é examinado e ENTENDIDO

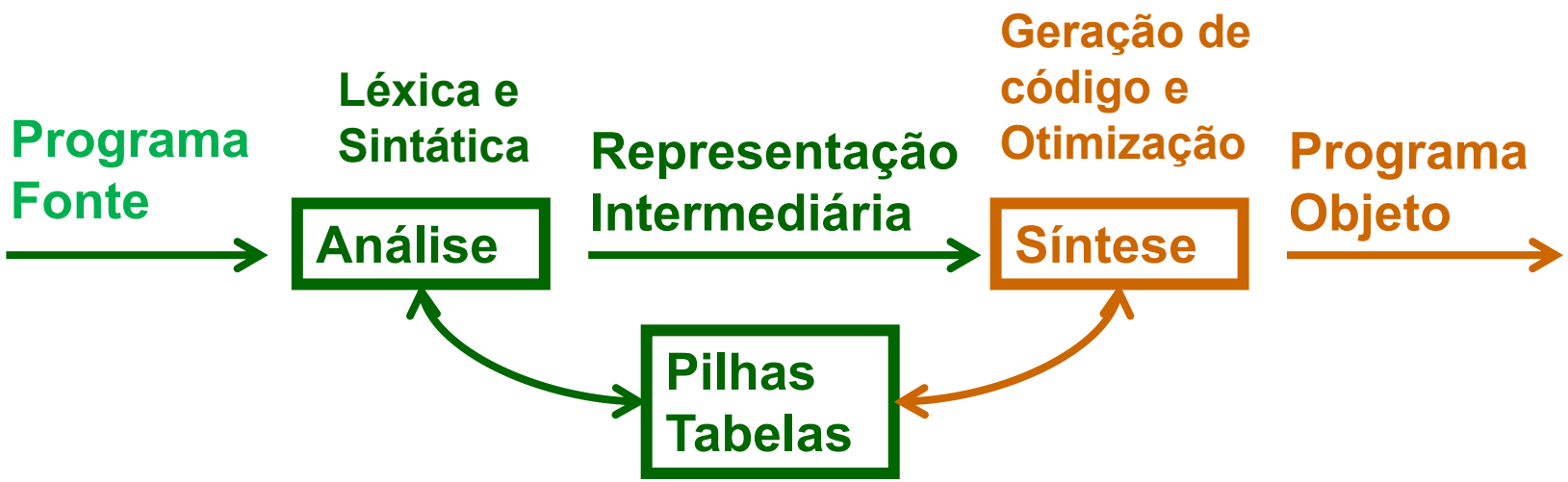
Síntese: (geração de código) - o texto de saída é gerado

Essas tarefas podem ser realizadas em paralelo ou sequencialmente

Em paralelo: cada **comando** é **analisado** e seu **código gerado de imediato**

Em sequência: cada **unidade do programa** é completamente **analisada** e depois o **código é gerado**

A estrutura global do compilador:

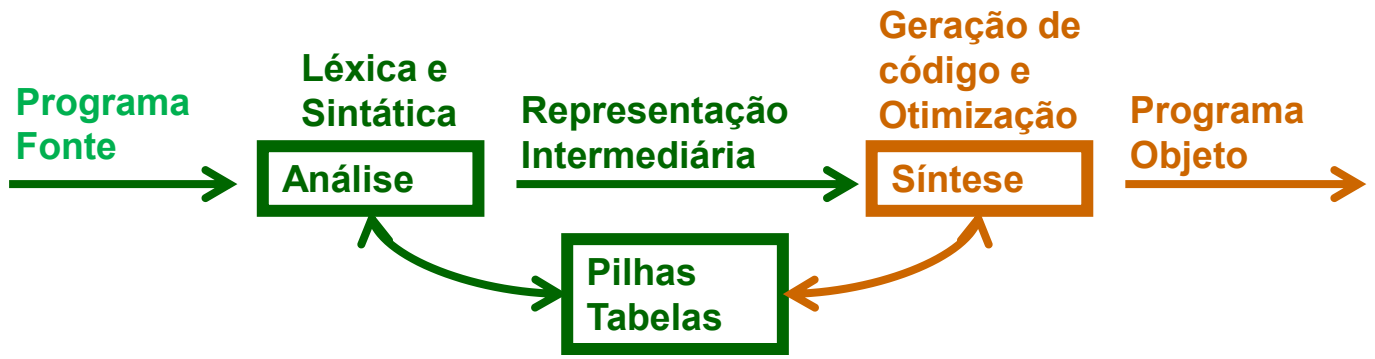


Representação intermediária:

assume a forma de **um programa em uma linguagem intermediária**
facilita a **tradução para a linguagem objeto** e
deve conter a **informação necessária** para a **geração do código objeto**

A facilidade para auxiliar a tradução se deve
às suas estruturas de dados,
que permitem acessos eficientes a todas as informações

A estrutura global do compilador:



Estruturas:

Tabelas: forma mais comum, a TABELA DE SÍMBOLOS

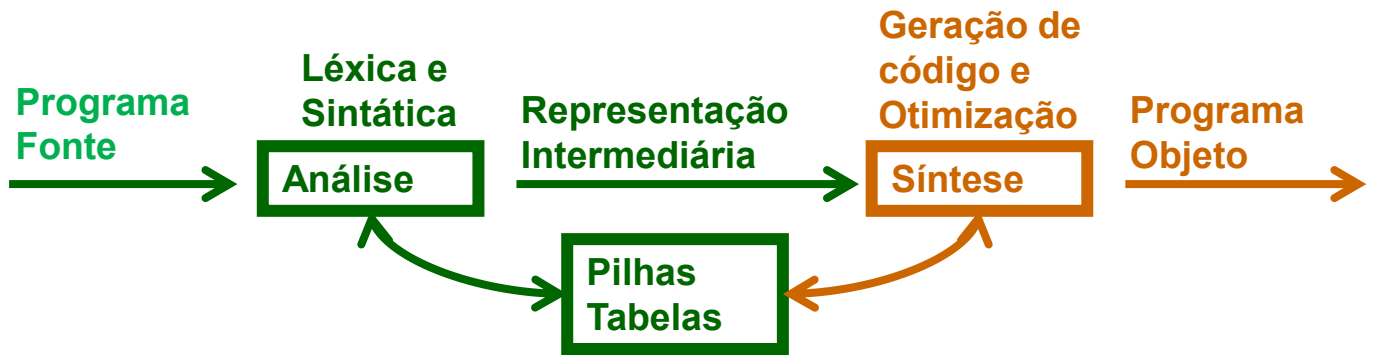
Cada **entrada da tabela** corresponde a um **identificador (símbolo)** usado no **programa** a ser compilado

Nessa entrada da tabela são guardados:

- o identificador,
- sua natureza (variável, constante, função, ou procedimento),
- seu tipo,
- seu endereço,
- o espaço a ser alocado, etc.

Capítulo 1 – Compiladores – Introdução

A estrutura global do compilador:



Estruturas:

Representação Intermediária - Pode ser concebida para que a ANÁLISE (front-end) fique INDEPENDENTE da SÍNTESE (back-end)

Comportamento desses módulos:

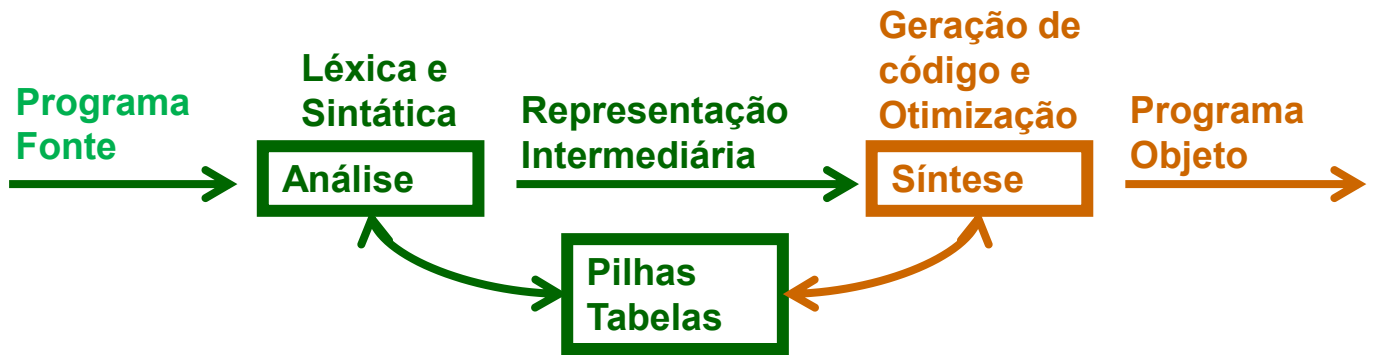
-front-end e back-end comunicam-se apenas via representação intermediária;

-front-end depende exclusivamente da linguagem fonte, i.e, independe da linguagem objeto ou da máquina de destino;

- back-end depende exclusivamente da linguagem objeto.

Capítulo 1 – Compiladores – Introdução

A estrutura global do compilador:



Representação Intermediária - Pode ser concebida para que a ANÁLISE (front-end) fique INDEPENDENTE da SÍNTESE (back-end)

Essa concepção necessita de uma linguagem intermediária para a qual devem convergir todas as traduções de programas em linguagens diferentes

Estas traduções podem ser aproveitadas em máquinas diferentes

Se tivermos m linguagens e n máquinas, em vez de fazermos m X n compiladores podemos fazer m front-ends e n back-ends.

Tradução dirigida por sintaxe - técnica mais usada hoje para compilação

Em cada regra gramatical da linguagem são agregadas ações correspondentes para o código intermediário

Por exemplo:
A regra de expressão na gramática, com operação aritmética de soma, dirige a tradução para uma ação de soma, envolvendo os seus operandos.

```
case 21:  /* E1 -> E2 + T */{
T = pop ();
pop (); // despreza o token +
E2 = pop ();
n = naoTerm [regra];
ultEstado = pilha [ip].estado;
push (n, Goto [ultEstado] [n]);
pilha [ip].loc = temp ();
sprintf(linha[PROX-1],
"%d %s %d %s %d\n", pilha[ip].loc,":=",
E2.loc, " + ", T.loc );//cod. Intermed.
PROX ++;
break;
};
```

21 é o estado no autômato que tem a redução da regra E -> E + T
Como é de baixo para cima, são dados três pops, o token de sinal é desprezado, porque o compilador sabe que este case trata a soma, os outros símbolos são guardados, porque em suas estruturas estão guardadas os endereços de memória dos operandos (expressões que podem ser ctes, variáveis ou temp de expressões)

Análise sintática – associada à forma

descreve os aspectos relativos à forma de construção de programas na linguagem.

Análise semântica – associada ao significado

descreve o que acontece na execução do programa.

A fase de análise se divide em 3:

1. **léxica**
2. **sintática**
3. **semântica**

Análise léxica – primeira análise, sendo bem mais simples que as outras;

Análise sintática – reconhece a estrutura global do programa

Análise semântica – encarrega-se da verificação das regras restantes, por exemplo, a verificação de tipo.

Ferramentas auxiliam a construção do **compilador** a partir da **especificação**

Ferramentas mais comuns:
construtores de análise léxica e sintática

Elas geram **módulos** até mais eficientes dos que os **compiladores** feitos a mão

Se a **análise semântica** puder ser especificada, é possível construir automaticamente um **compilador**, para esta especificação, com algum ajuste para melhorar o desempenho.

Separação e identificação dos elementos componentes (lexemas ou tokens) do fonte;

Eliminação dos elementos "decorativos" do programa:
espaços, marcas de formatação e comentários

Após a análise léxica, os itens -
identificadores, operadores, delimitadores, palavras reservadas -
ficam identificados, geralmente, por duas informações:
-Um código numérico
-A cadeia correspondente ao fonte

Por exemplo,
considere o trecho
de programa Pascal:

```
if x>0
 then modx := x
 else modx :=(-x)
```

Após a análise léxica,
a sequência de tokens identificada é:

tipo do token	valor do token
palavra reservada if	if
Identificador	x
operador maior	>
literal numérico	0
palavra reservada then	then
identificador	modx
operador de atribuição	:=
identificador	x
palavra reservada else	else
identificador	modx
operador de atribuição	:=
delimitador abre parêntese	(
operador menos unário	-
identificador	x
delimitador fecha parêntese	)

Implementação de analisador léxico (scanner) é baseada em um autômato finito para reconhecer cada construção

Lembrando que qualquer expressão regular pode ser transformada em um autômato finito automaticamente,

a **expressão regular** para um **identificador** pode ser

```
letra.(letra V digito V sublinhado)*
```

Em que **`letra`**, **`digito`** e **`sublinhado`** representam conjuntos (ou classes) de **`caracteres`**:

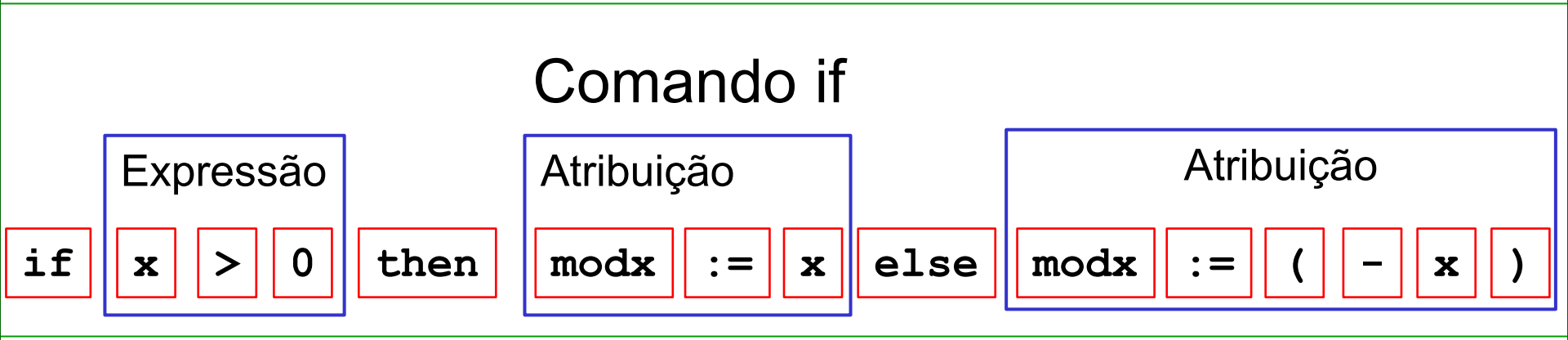
```
letra = { 'A', ...., 'Z', 'a', ... 'z' }
digito = { '0', ..., '9' }
sublinhado = { '_' }
```

Reconhece a estrutura global do programa, verificando se os:

Declarações
Comandos
Expressões
Etc

Estão de acordo com as regras da gramática

No exemplo anterior, o analisador sintático reconhece o trecho do fonte:



Capítulo 1 – Análise Semântica

Trata os aspectos Sensíveis ao Contexto da Sintaxe das Linguagens

Por exemplo:

Não é possível representar numa GLC uma regra, como a seguinte:
"Todo o identificador deve ser declarado antes de ser referenciado"

Mas é possível adicionar mecanismos ao analisador, para verificar esta condição, fazendo uso de dados auxiliares, como por exemplo, a tabela de símbolos.

Na forma "DIRIGIDA PELA SINTAXE", quer dizer, a cada conclusão de regra, se associa uma ação semântica à regra sendo concluída,

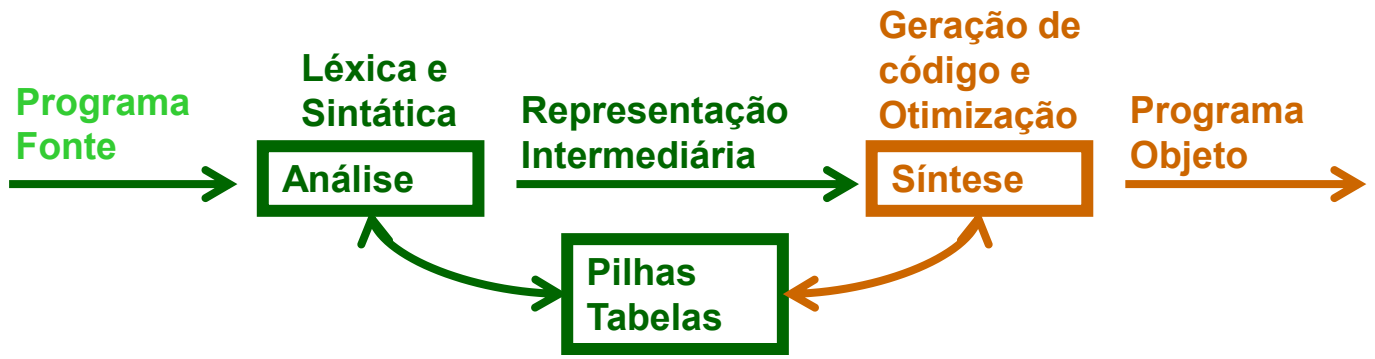
quando a análise de uma regra de declaração é concluída,
a ação semântica associada inclui os identificadores na tabela de símbolos.

Quando algum identificador é encontrado em outros trechos do programa, é verificada a sua presença na tabela de símbolos.

Pode-se observar que a análise semântica e a análise sintática são realizadas em combinação, conjuntamente, aparecendo no mesmo trecho de código do compilador.

Capítulo 1 –Geração de código e Otimização dependente de máquina

A estrutura global do compilador:



O processo de geração de código dependerá da arquitetura da máquina, na qual o programa deverá ser executado

Ex. Código gerado para uma máquina de um operando:

fonte	intermediário	código gerado	código otimizado
x := a +b * c	t1:= b * c	LOAD b	LOAD b
		MULT c	MULT c
		STORE t1	ADD a
	t2:= a + t1	LOAD a	STORE x
		ADD t1	
		STORE t2	
	x := t2	LOAD t2	
		STORE x	

A transformação acontece ainda no código intermediário

Exemplo:

Fonte	intermediário	otimizado
w := (a + b) + c;	t1 := a + b	t1:= a+b
	t2 := t1 + c	t2:=t1+c
	w := t2	w:=t2
x := (a + b)* d;	t3 := a + b	t4 := t1 * d
	t4 := t3 * d	x := t4
	x := t4	
y := (a + b) + c;	t5 := a + b	y := t2
	t6 := t5 + c	
	y := t6	