Compiladores

Fabio Lubacheski
fabio.lubacheski@mackenzie.br
http://lattes.cnpq.br/9894811024725114

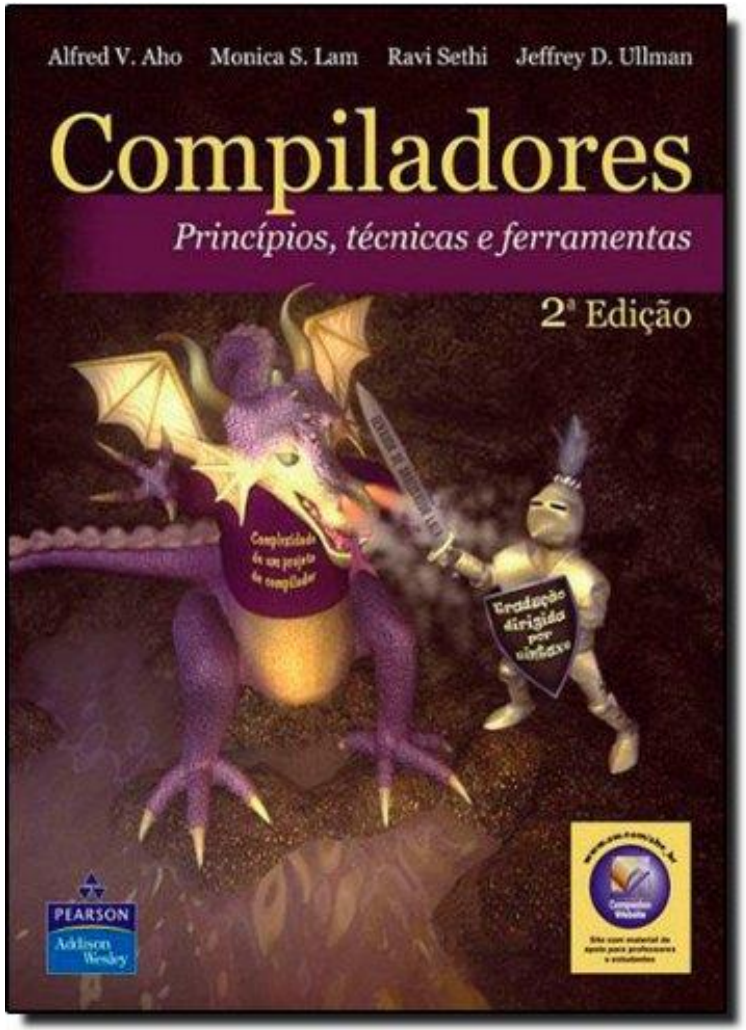
Plano de Ensino

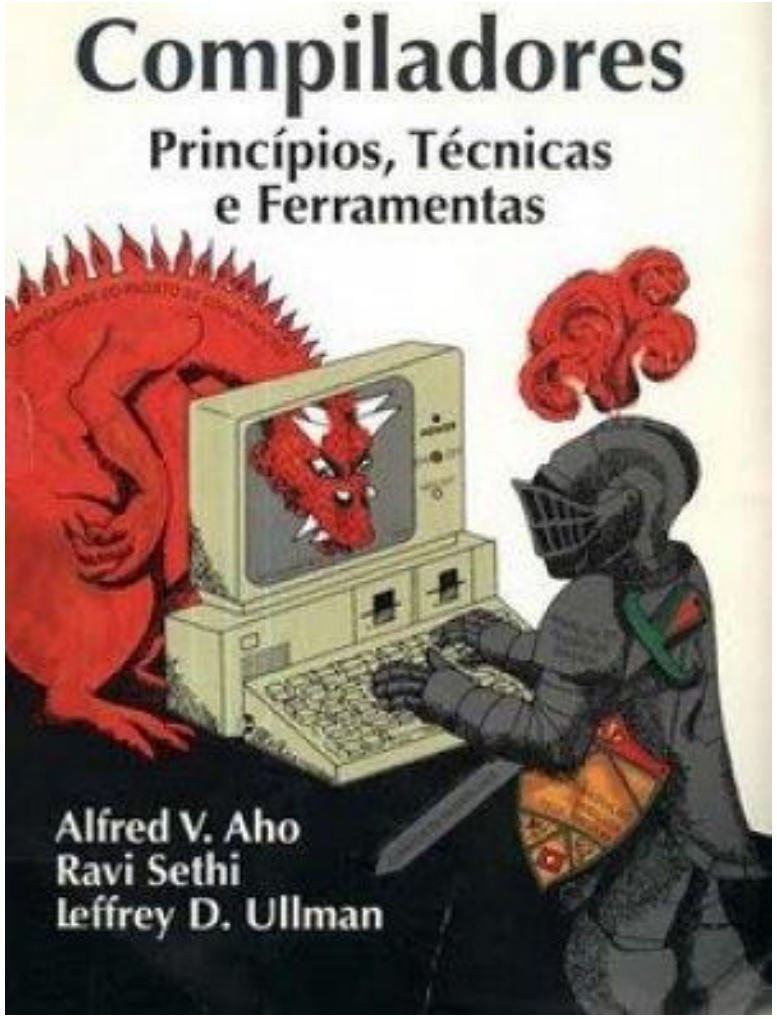
UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática

UNIDADE - FACULDADE DE COMPUTAÇÃO E INFORMÁTICA		
CURSO: CIÊNCIA DA COMPUTAÇÃO		**NÚCLEO TEMÁTICO:** ALGORITMOS E PROGRAMAÇÃO
DISCIPLINA – COMPILADORES		**CÓDIGO DA DISCIPLINA** ENEX01047
PROFESSOR(ES) FABIO APARECIDO GAMARRA LUBACHESKI	**DRT** 1146330	**ETAPA** 6º
CARGA HORÁRIA 4h/a (2 teoria | 0 laboratório | 2 EAD)		**SEMESTRE LETIVO** 2024/2
EMENTA Estudo das fases dos processos de compilação e linguagens. Estudo dos esquemas de análise léxica, análise sintática, análise semântica, geração de código intermediário, alocação de registradores e geração de código Assembly. Estudo comparativo de ferramentas de geração de compiladores.		

Livros e referências

Livro do Dragão





Livros e referências

Aulas do Prof. Tomasz Kowaltowski em:

https://sites.google.com/unicamp.br/ic-mo403-mc900/main

E também o livro disponível em:

https://drive.google.com/file/d/1DyqeBUgayQpjh1yaHziQgm8WCE7u7ZaN/view

Compiladores

- Por que estudar compiladores ?

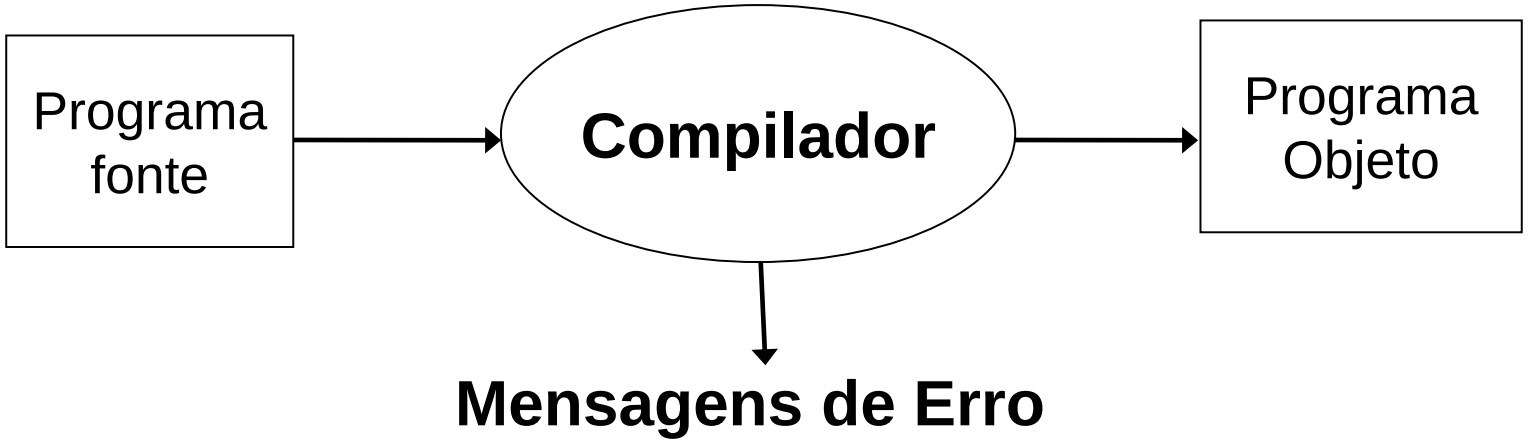
  Os princípios e técnicas da construção de compiladores são usadas muitas vezes na carreira de um cientista da computação.

  Permite ao cientista da computação entender melhor os erros de desenvolvimento em linguagens de programação, proporcionar uma experiência prática no desenvolvimento de uma linguagem de programação, incentivar a criatividade, inovação, novas abstrações no desenvolvimento de software.

- Pré-requisitos:
  - Linguagens de Programação;
  - Linguagens Formais e Autômatos;
  - Sistemas Operacionais;
  - Arquitetura de Computadores; e
  - Algoritmos e Estrutura de Dados.

Definicões iniciais

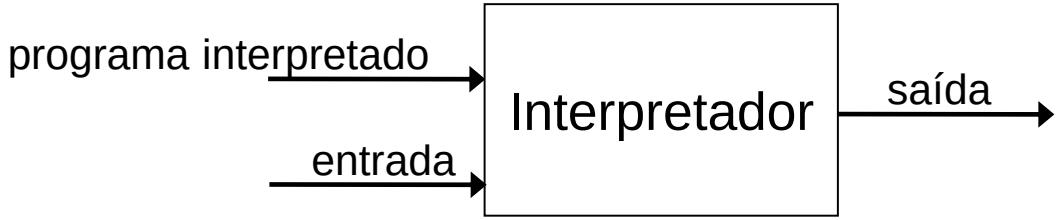
Um compilador é um **processador de linguagem**, cuja função é receber como **entrada** um programa escrito em uma linguagem de programação (linguagem fonte) e converte para um programa equivalente em outra linguagem (linguagem objeto | linguagem alvo).



Definicões iniciais

Se o programa objeto for uma programa em uma **linguagem de máquina** executável, poderá ser chamado pelo usuário para processar **entradas** e produzir **saída**.

Um **interpretador** é outro tipo de **processador de linguagem**, mas não produz um **programa objeto** como saída, um interpretador executa diretamente as operações especificadas no **programa fonte** sobre as entradas fornecidas pelo usuário.



Programas relacionados aos Compiladores

- **Interpretadores**:

  Um interpretador é um processador de linguagens, assim como um compilador. A diferença é que o interpretador executa o programa-fonte de imediato, em vez de gerar um código-objeto. Os interpretadores compartilham muitas operações com os compiladores, podendo inclusive existir processadores híbridos.

  Exemplo JVM??

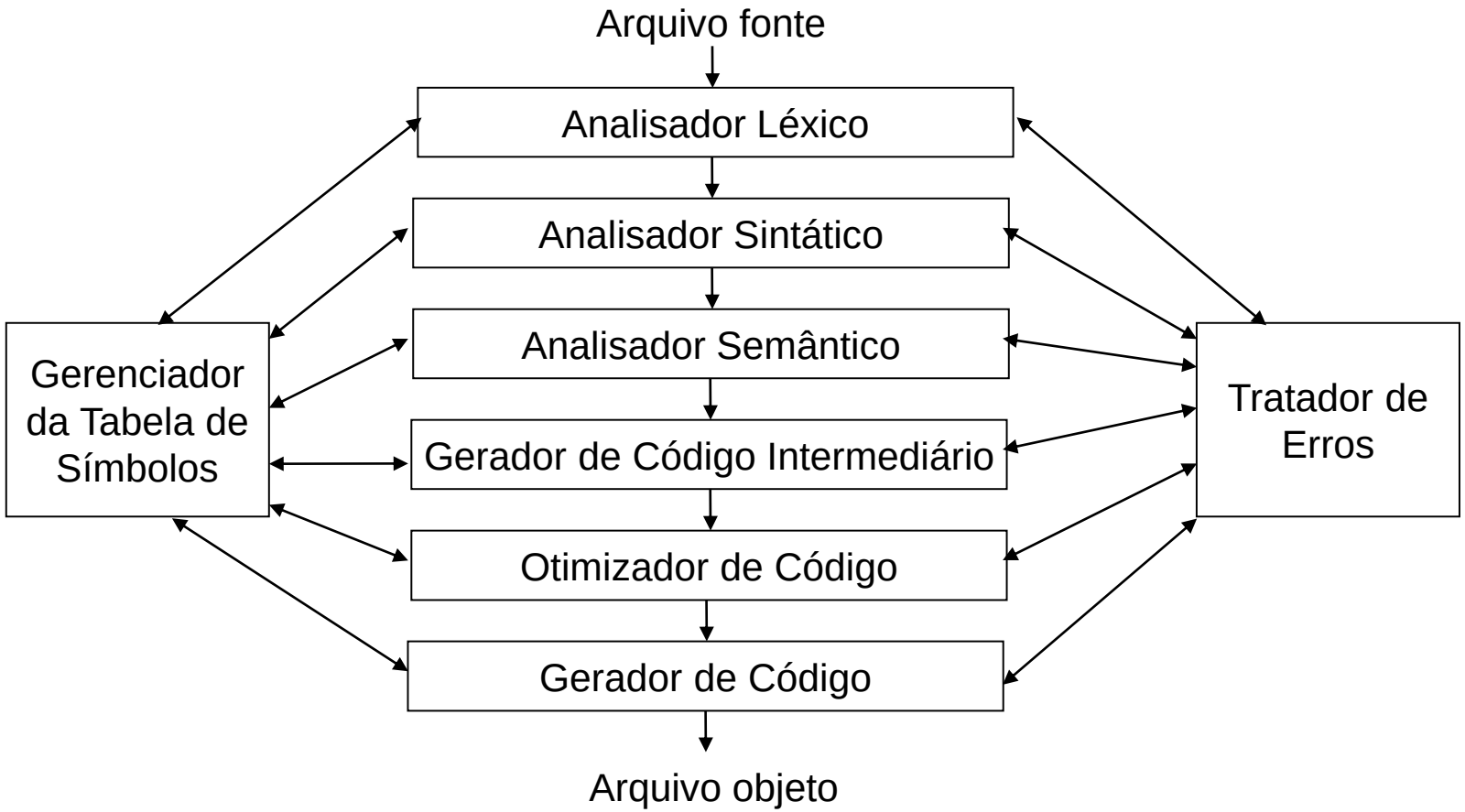
- **Pré-processadores:**

  É um programa separado que é chamado pelo compilador antes do início do processo de compilação. Ele pode apagar comentários, incluir arquivos e executar substituição de macros dentre outras coisas.

- **Depuradores:**

  É utilizado para determinar erros de execução em um programa compilado, e costuma ser integrado com um IDE. Para efetuar suas funções, o depurador precisa receber informação simbólica apropriada do compilador.

Fases do Compilador



Fases do Compilador

- **Gerenciador da tabela de símbolos**

  Sua função é registrar os identificadores usados no programa fonte e coletar informações sobre os seus atributos. A tabela de símbolos é um estrutura de dados com um registro para cada identificador, com campos contendo os seus atributos, além disso essa estrutura deve permitir uma recuperação bastante eficiente da informação armazenada.

- **Tratador de erros**

  Cada fase do compilador pode conter erros, que precisam ser tratados de tal forma que a compilação possa continuar.

Analisador léxico

- lê o arquivo fonte, agrupando os caracteres e classificando em átomos;
- Comentários, espaços e outros caracteres de controle são ignorados;
- Inicia a construção da tabela de símbolos.
- Repassa os tokens (códigos internos) para o analisador sintático; e
- Normalmente é implementado como uma subrotina que funciona sob o comando auxiliar do analisador sintático.

Imagine que o analisador léxico leia a seguinte linha em um arquivo fonte.

```
double montante = deposito_inicial+taxa_de_juros*60;
```

Analisador léxico

- Linha no arquivo fonte

  `double montante = deposito_inicial+taxa_de_juros*60;`

- Os **tokens** poderiam ser agrupados da seguinte forma:

  `double` (palavra reservada): **`double`**
  identificador (`id1`): **`montante`**
  símbolo atribuição: **`=`**
  identificador (`id2`): **`deposito_inicial`**
  operador adição: **`+`**
  identificador (`id3`): **`taxa_de_juros`**
  operador multiplicação: **`*`**
  número: **`60`**

- Os espaços desses tokens são eliminados pelo analisador léxico.

Analisador léxico

Certos tokens serão enriquecidos por um valor léxico, por exemplo o identificador **`taxa_de_juros`**, além de ser gerado um token identificador, ele também é inserido na tabela de símbolos na entrada **`id3`**:

`Tabela de símbolos`

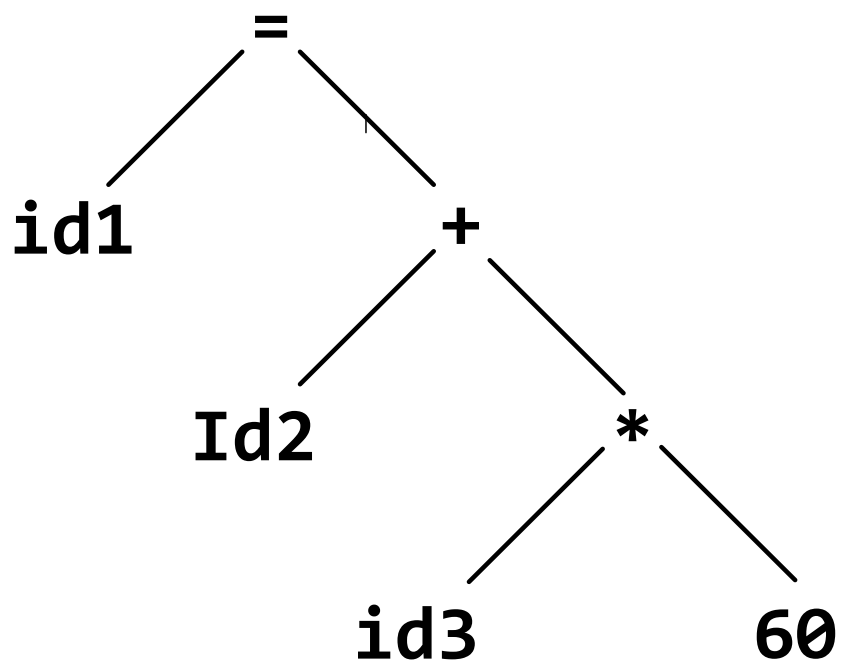
	identificador	**categoria**	**tipo**	**….**
id1	montante	VAR	double	
id2	deposito_inicial	VAR	double	
id3	taxa_de_juros	VAR	double	
….	….	….	….	….

Analisador léxico para Analisador sintático

O **analisador léxico** repassa os **tokens** para o **analisador sintático** e a expressão da entrada convertida em uma representação interna com as entradas dos identificadores na tabela de símbolos:

```
id1 = id2+id3*60
```

A partir da entrada do analisado léxico o **analisador sintático** produz uma **árvore sintática** com a expressão.



Analisador sintático

O **analisador sintático** verifica se a estrutura gramatical do programa está correta. Isto é, se essa estrutura foi formada usando as regras gramaticais da linguagem;

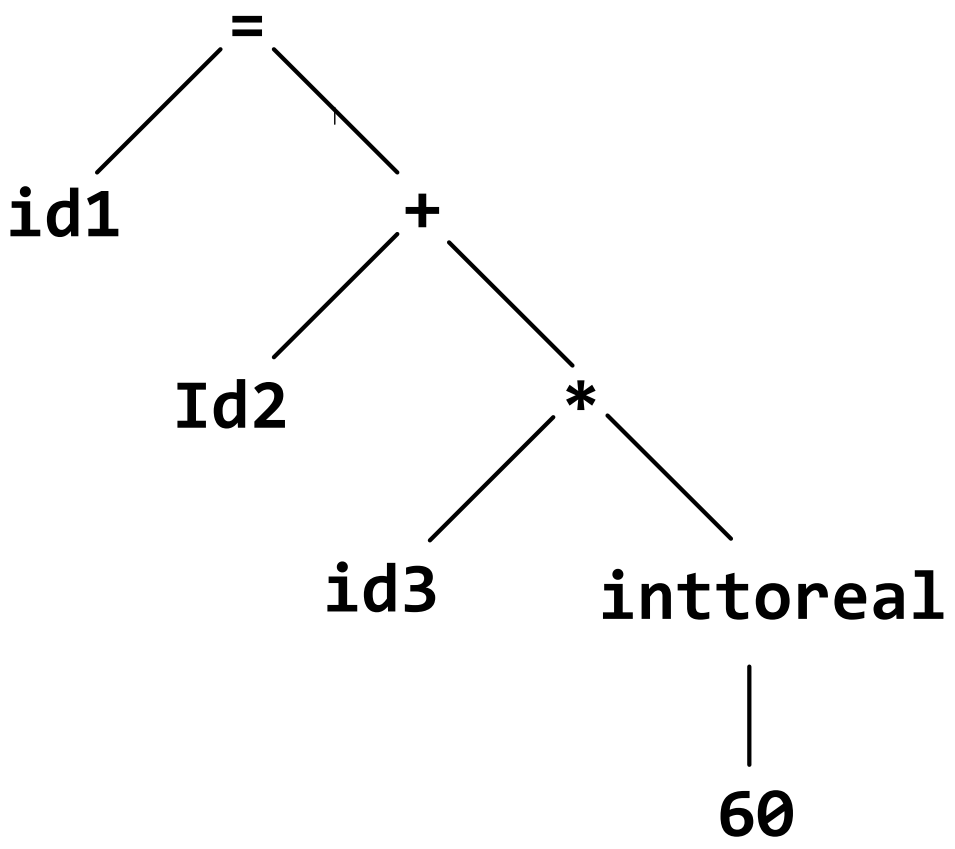
árvore, chamada **árvore sintática ou árvore de derivação**, é usada para detecção de erros de sintaxe; e por fim

O analisador sintático opera conjuntamente com o **analisador semântico**.

Analisador semântico

Verifica se as construções sintaticamente corretas possuem significado lógico dentro da linguagem;

Analisador semântico faz conversões que se fazem necessárias, e permitidas, para dar significado a uma sentença fazendo alterações na árvore sintática.



Gerador de código intermediário

Efetua o mapeamento do árvore sintática, enriquecida com as informações semânticas, para a linguagem intermediária;

Geração de código é feita para uma **linguagem intermediária** de nível próximo ao código objeto. Na maioria das vezes temos um **código de 3 endereços**;

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

Não há preocupação com a escolha dos registradores oferecidas pela máquina onde o programa será executado.

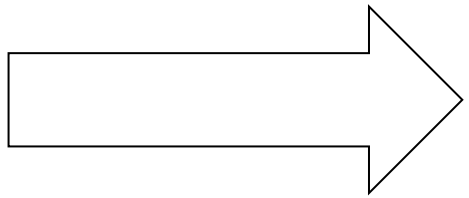
Otimizador de código

Otimiza o código intermediário em termos de velocidade de execução e espaço de memória; e

Após a otimização o programa alvo deve continuar funcionando corretamente

Exemplos de otimizações: Atribuições redundantes, suprimir subexpressões comuns, melhor utilização dos registradores e etc...

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```



```
temp1 = id3 * 60.0
id1 = id2 + temp1
```

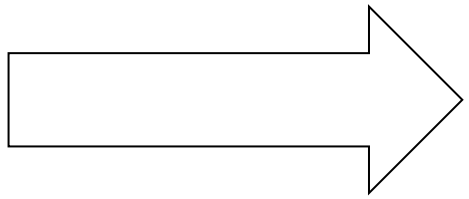
Gerador de código objeto

Geração do código alvo, normalmente um código de máquina;

As localizações de memória são selecionadas para cada uma das variáveis usadas pelo programa

Instruções intermediárias são traduzidas numa sequência de instruções de máquina que realizam a mesma tarefa.

```
temp1 = id3 * 60.0
id1 = id2 + temp1
```



```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

Para saber mais

- **Leia os capítulo 1 e 2 do livro** AHO, A.V., LAM, M.S., SETHI, R., ULLMAN, J.D. **Compilers: Principles, Techniques and Tools**. 2.ed. New York: Addison-Wesley, 2006.

Exercícios – Questões ENADE EngComp – 2008 – questão 52

A identificação e o tratamento de erros em programas de computador estão entre as tarefas dos compiladores. Os erros de um programa podem ter variados tipos e precisam ser identificados e tratados em diferentes fases da compilação. Considere uma linguagem de programação que exige que as variáveis manipuladas por seus programas sejam previamente declaradas, não podendo haver duplicidade de identificadores para variáveis em um mesmo escopo. Considere, ainda, que a sintaxe dessa linguagem tenha sido definida por meio de uma gramática livre de contexto e as produções seguintes definam a forma das declarações de variáveis em seus programas.

$D \rightarrow TL; \mid TL; D$

$T \rightarrow int \mid real \mid char$

$L \rightarrow id \mid id,L$

Exercícios – Questões ENADE EngComp – 2008 – questão 52

Considere os exemplos de sentenças — **I** e **II** — a seguir, com a indicação — entre os delimitadores `/*` e `*/` — de diferentes tipos de erros.

I `int: a, b; /* dois pontos após a palavra int */`

II `int a,b; real a; /* declaração dupla da variável a */`

A partir dessas informações, assinale a opção correta.

A) identificação e a comunicação do erro em qualquer uma das sentenças são funções do analisador léxico.

B) O compilador não tem meios para identificar e relatar erros como o da sentença **I**.

C) A identificação e a comunicação do erro na sentença **I** são funções da geração de código intermediário.

D) A identificação e a comunicação do erro na sentença **II** são funções do analisador léxico.

E) A identificação e a comunicação do erro na sentença **II** são funções da análise semântica.

Exercícios – Questões ENADE EngComp – 2017 – questão 20

Um compilador transforma uma linguagem, em geral textual, em outra linguagem. Um dos tipos de linguagens que um compilador pode transformar são as linguagens regulares, que podem ser descritas utilizando-se expressões regulares compostas por símbolos isolados agrupados com operadores `*` e `U` e organizadas com auxílio de parênteses.

Nesse contexto, avalie as afirmações a seguir.

I. A palavra `10010100` pertence à linguagem representada por `(100*)*`.

II. A palavra `10010` pertence à linguagem representada por `(1(10)*0)*`.

III. Existe somente uma expressão regular para representar uma linguagem regular.

É correto o que se afirma em:

A) I, apenas.

B) II, apenas.

C) I e III, apenas.

D) II e III, apenas.

E) I, II e III.

Exercícios

1) Escreva um programa na **linguagem C** que leia um arquivo texto e contabilize a quantidade ocorrência das letras **`A..Za..z`**, dígitos **`0..9`** e espaços em branco no arquivo texto. Ao final o seu programa deve imprimir uma listagem **quantidade da letras e dígitos em ordem crescente**, **quantidade de espaços em branco** e o **número de linhas no arquivo fonte**. Caso para determinado símbolo não tenha a ocorrência não é necessário imprimir a quantidade **`0`**, também **não é necessário** contar caracteres especiais com acento, como por exemplo **à** ou **ç**.
2) Escreva um programa na **linguagem C** que leia um arquivo texto e contabilize a quantidade de palavras nas linhas de um arquivo texto, a cada linha o seu programa deve informar a **quantidade de palavras encontradas**. Para esse exercício considere que as palavras serão formadas somente por letras maiúsculas e minúsculas sem caracteres especiais, como por exemplo **à** ou **ç**, considere também que as palavras são separadas por espaços em branco, virgula, ponto e ponto e vírgula.

Desafio – Como seria programar em uma linguagem sem variáveis?

- Suponha o seguinte problema:

  Escreva uma função que recebe um ponteiro para vetor de caractere por parâmetro (somente uma ponteiro), veja a declaração da função:

```
int Impar_a_Par_b(char *entrada);
```

  A sua função retorna `1` se a quantidade de **a´s** for **ímpar** e a quantidade **b´s** for **par**, e `0` caso contrário.

  Restrição importante: Na implementação da sua função você **não pode usar mais nenhuma variável** (global, local ou parâmetro a mais), somente o parâmetro de entrada.

Desafio – Como seria programar em uma linguagem sem variáveis?

O vetor de caractere que foi passado por parâmetro só possuí os caracteres **a** e **b**, por exemplo:

```
int main(void){
  char *entrada = “ababbaaba”;
  Printf(“reposta: %d”, Impar_a_Par_b(entrada));
}
```

Fim