UNIVERSIDADE FEDERAL DE GOIÁS – UFG

CAMPUS CATALÃO – CaC

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – DCC

Bacharelado em Ciência da Computação

Projeto Final de Curso

Estudo de um compilador sob a perspectiva de um sistema online de programação

Autor: Nélio Carneiro Júnior

Orientador: Dr. Thiago Jabur

Catalão - 2011

Nélio Carneiro Júnior

Estudo de um compilador sob a perspectiva de um sistema online de programação

Monografia apresentada ao Curso de
Bacharelado em Ciência da Computação da
Universidade Federal de Goiás Campus Catalão
como requisito parcial para obtenção do título de
Bacharel em Ciência da Computação

Área de Concentração: Compiladores
Orientador: Dr. Thiago Jabur

Catalão - 2011

Carneiro, Nélio

Estudo de um compilador sob a perspectiva de um sistema online de programação/Dr. Thiago Jabur- Catalão - 2011

Número de paginas: 5

Projeto Final de Curso (Bacharelado) Universidade Federal de Goiás, Campus Catalão, Curso de Bacharelado em Ciência da Computação, 2011.

Palavras-Chave: 1. Compilador. 2. Sistemas Web. 3. Linguagem de Programação

Nélio Carneiro Júnior

Estudo de um compilador sob a perspectiva de um sistema online de programação

Monografia apresentada e aprovada em _____ de _____
Pela Banca Examinadora constituída pelos professores.

_____

Dr. Thiago Jabur – Presidente da Banca

_____

Professor 1

_____

Professor 2

Dedico este trabalho

AGRADECIMENTOS

Por fim, agradeço a todos que de forma direta ou indireta que cotribuíram para a minha formação.

"Os desafios.

Sulamita

Sumário

Lista de Figuras

Introdução

É evidente o crescente número de usuários conectados e dependentes da internet ao longo do tempo. Percebe-se também que estes usuários, estão cada vez mais necessitados de que suas ferramentas, antes usadas em seus desktops, fiquem disponíveis online, de prontidão, sempre que se fizer necessário, mesmo estando conectados longe de casa. Fez-se então necessário que o desenvolvimento de softwares caminhasse neste mesmo sentido. [carece de fontes]

Um aspecto interessante deste processo a ser analisado são as aplicações que dependem de um compilador, ou seja, programas que quando instalados na máquina sempre fazem uso de um compilador ou então, são eles mesmos, compiladores. Um exemplo deste tipo de software seria um compilador para Portugol. Como iria se comportar tal compilador ao ser usado na internet? Como se daria a implementação deste compilador? Quais aspectos a serem tratados? E como seria o sistema na web que necessitasse deste compilador?

O objetivo principal deste trabalho é analisar o comportamento de um compilador feito para a linguagem Portugol[fontes], trabalhando sob um sistema de progamação na Web. A proposta é fazer um estudo detalhado da implementação deste compilador, tendo assim uma base sólida para o desenvolvimento do mesmo, visando na prática tirar conclusões à respeito das técnicas utilizadas.

Este trabalho também tem como objetivo desenvolver uma aplicação Web que irá utilizar o compilador. A aplicação será um editor o qual o usuário fará seus programas em Portugol, tendo de imediato a resposta do compilador. A intenção é analisar como irá se comportar o compilador, bem como, verificar a viabilidade de uso de tal aplicação, ao se implementar utilizando as técnicas descritas ao longo deste trabalho. A aplicação terá um caráter educativo, no qual o usuário fará programas e, ele e seus trabalhos, poderão ser acompanhados por um instrutor, visando assim verificar os passos realizados pelo usuário, bem como a maneira pela qual o compilador tem sido usado.

No capítulo 1 são abordados os conceitos de compiladores, as técnicas utilizadas na criação do compilador e como se dará o seu funcionamento diante uma aplicação Web.

No capítulo 2 é feita uma apresentação da linguagem do compilador: o Portugol. Suas características e o embasamento para a criação do compilador baseado nesta linguagem também fará parte deste capítulo

No capítulo 3 pode-se observar um levantamento sobre o estado da arte dos compiladores e como tem sido o uso destes em aplicações na Web. Neste também irei citar casos em que editores/compiladores foram colocados na Web.

No capítulo 4 é discorrido sobre a metodologia proposta para o desenvolvimento do protótipo de editor/compilador Web.

No capítulo 5 é abordado todo o processo de testes, além de ser realizado uma análise dos resultados. Por fim há uma conclusão do trabalho realizado, levantando os objetivos alcançados, pontos positivos, pontos negativos, dificuldades e trabalhos futuros.

Ainda é possível encontrar o código fonte do compilador e da aplicação Web criada, além de toda a documentação de testes elaborada no apêndice desta monografia.

Capítulo 1

Compiladores

1.1 Introdução

Criado por volta dos anos 50, o nome Compilador se refere ao processo de composição de um programa através da reunião de várias rotinas de bibliotecas. O processo de tradução (de uma linguagem fonte para uma linguagem objeto), considerado hoje a função central de um compilador, era então conhecido como programação automática[Rangel, 1999]

Definido em [AHO, 1995], um compilador é um programa que lê outro programa escrito em uma linguagem — a linguagem de origem — e o traduz em um programa equivalente em outra linguagem — a linguagem de destino. Como uma importante parte no processo de tradução, o compilador reporta ao seu usuário a presença de erros no programa origem.

Ao longo dos anos 50, os compiladores foram considerados programas notoriamente difíceis de escrever. O primeiro compilador Fortran, por exemplo, consumiu 18-homens ano para implementar[Backus, 1957]. Desde então, foram descobertas técnicas sistemáticas para o tratamento de muitas das mais importantes tarefas desenvolvidas por um compilador.

A variedade de compiladores nos dias de hoje é muito grande. Existem inúmeras linguagens fontes, as quais poderiam ser citadas em várias páginas deste trabalho. Isso se deve principalmente ao fato de que com o aumento do uso dos computadores, aumentou também, as necessidades de cada indivíduo, sendo estas específicas, exigindo por sua vez linguagens de programação diferentes. Este processo — juntamente com a evolução da tecnologia de desenvolvimento de compiladores — levou à criação de várias técnicas diferentes para a construção de um compilador, ou seja, passou a existir diferentes maneiras de se implementar um compilador. No entanto, a despeito dessa aparente complexidade, as tarefas básicas que qualquer compilador precisa realizar são essencialmente as mesmas.

A grande maioria dos compiladores de hoje fazem uso da técnica chamada: *tradução*

dirigida pela sintaxe. Nesta técnica as regras de contrução do programa fonte são utilizadas para guiar todo o processo de compilação. Algumas das técnicas mais antigas utilizadas na contrução dos primeiros compiladores (da linguagem Fortran) pode ser obtido em [Rosen, 1967].

1.2 Modelo de Compilação de Análise e Síntese

Ainda segundo [Rangel, 1999], existem duas tarefas triviais a serem executadas por um compilador nesse processo de tradução:

- *análise*, em que o texto de entrada (na linguagem fonte) é examinado, verificado e compreendido
- *síntese*, ou *geração de código*, em que o texto de saída (na linguagem objeto) é gerado, de forma a corresponder ao texto de entrada.

Em [Aho, 1995], *análise* é colocada como uma tarefa que divide o programa fonte nas partes constituintes e cria uma representação intermediária do mesmo. E *síntese* constrói o programa alvo desejado, a partir da representação intermediária.

Geralmente, pensamos nessas tarefas como fases que ocorram durante o processo de compilação. No entanto, não se faz totalmente necessário que a análise de todo o programa seja realizada antes que o primeiro trecho de código objeto seja gerado. Ou seja, estas duas fases podem ser intercaladas. Por exemplo, o compilador pode analisar cada comando do programa de entrada e então gerar de imediato o código de saída correspondente ao respectivo comando. Ou ainda, o compilador pode esperar pelo fim da análise de cada bloco de comando — ou unidade de rotina (rotina, procedimentos, funções) — para então gerar o código correspondente ao bloco. Para aproveitar melhor a memória durante a execução, compiladores costumavam ser divididos em várias etapas, executados em sequência. Cada etapa constitui uma parte do processo de tradução, transformando assim o código fonte em alguma estrutura intermediária adequada, cada vez mais próxima do código objeto final.

É natural que a análise retorne como resultado uma representação do programa fonte que contenha informação necessária para a geração do programa objeto que o corresponda. Quase sempre, essa representação (conhecida como *representação intermediária*[Rangel, 1999]) tem como complemento tabelas que contêm informações adicionais sobre o programa fonte. Pode ter casos em que a representação intermediária toma a forma de um programa em uma *linguagem intermediária*, deixando assim mais fácil a tradução para a linguagem objeto desejada.

Não importando a maneira pela qual se toma a representação intermediária, ela tem de conter necessariamente toda a informação para a geração do código objeto. Uma das características da representação intermediária é que as estruturas de dados implementadas devem dar garantia de acesso eficiente as informações.

Imagem - Rangel[1999]

Segundo [Rangel, 1999], uma das formas mais comuns de tabela utilizada nessa representação intermediária é a *tabela de símbolos*, em que se guarda para cada identificador(*símbolo*) usado no programa as informações correspondentes.

Há também um modelo possível em [Ullman, 1977], o qual se faz a separação total entre o *front-end*, encarregado da fase de análise, e o *back-end*, encarregado pela geração de código. Com isso tem-se que:

- front-end e back-end se comunicam apenas da representação intermediária;
- o front-end depende exclusivamente da linguagem fonte
- o back-end depende exclusivamente da linguagem objeto.

Essa idéia tem como objetivo simplificar a implementação de diferentes linguagens de programação para diferentes máquinas. Basta-se então escrever um front-end para cada linguagem e um back-end para cada máquina. Ou seja, se deseja implementar x linguagens para y máquinas, precisa-se fazer x front-ends e y back-ends. Este esquema se torna mais fácil de aplicar quando há semelhança entre as máquinas e o mesmo acontece com as linguagens.