README - Disciplina de Compiladores

Esta disciplina tem como objetivo fornecer uma compreensão profunda sobre os conceitos, técnicas e ferramentas envolvidas na construção de compiladores, desde as linguagens formais até a geração de código otimizado. Ao longo do curso, abordaremos diversos tópicos fundamentais para o desenvolvimento de compiladores modernos, com ênfase em suas várias fases e componentes.

Tópicos Abordados

1. Introdução a Compiladores

A introdução a compiladores apresenta os conceitos básicos sobre o papel de um compilador no processo de tradução de código-fonte de uma linguagem de alto nível para uma linguagem de baixo nível (geralmente código de máquina). Discute-se a importância dos compiladores para a criação de programas eficientes e os principais estágios que compõem o processo de compilação.

2. Linguagens Formais

As linguagens formais são um conjunto de regras para gerar frases válidas em uma linguagem. Elas são fundamentais para a definição da sintaxe de uma linguagem de programação e servem de base para os compiladores. O estudo das linguagens formais envolve o uso de gramáticas, como as gramáticas livres de contexto, que são usadas para descrever a estrutura de linguagens de programação.

3. Autômatos Finitos Determinísticos

Autômatos finitos determinísticos (AFD) são modelos matemáticos usados para reconhecer padrões e linguagens formais. Eles são fundamentais para a análise léxica de um compilador, onde são usados para identificar tokens em uma sequência de caracteres de entrada. Um AFD possui um número finito de estados e transições, com um único estado de transição para cada símbolo de entrada.

4. Expressões Regulares

Expressões regulares são uma forma compacta de descrever padrões de texto. Elas são usadas em compiladores para definir as regras de tokens, ou seja, os menores blocos de significado (como palavras-chave, operadores e identificadores) na linguagem de programação. Elas são processadas pelo analisador léxico para segmentar o código-fonte.

5. Organização de Computadores

A organização de computadores trata dos componentes internos de um computador e como eles interagem para executar programas. Em um compilador, entender a arquitetura do processador, a memória e o sistema de entrada/saída é essencial para gerar código eficiente. Este tópico é importante para o desenvolvimento de compiladores que produzam código otimizado para plataformas específicas.

6. Introdução à Compilação

A introdução à compilação oferece uma visão geral das etapas que um compilador deve executar para transformar um programa fonte em código executável. Essas etapas incluem análise léxica, análise sintática, análise semântica, otimização e geração de código.

7. Representações Intermediárias Gráficas

As representações intermediárias gráficas (GIR) são uma forma visual de representar o programa durante as etapas de compilação. Elas são usadas para analisar e otimizar o código de forma mais intuitiva, permitindo a visualização de fluxos de controle e dependências entre operações. Esse formato pode ajudar a compreender a execução e otimização do código.

8. Representações Intermediárias Lineares

Representações intermediárias lineares (IRL) são uma forma de representar o código de maneira sequencial, utilizando uma lista ou sequência de operações. Elas são geralmente mais compactas e eficientes em termos de processamento, além de serem mais fáceis de manipular por otimizadores e geradores de código.

9. Análise Léxica

A análise léxica é a primeira fase da compilação, responsável por dividir o código-fonte em tokens, que são as unidades mínimas de significado. O analisador léxico utiliza expressões regulares e autômatos finitos determinísticos para identificar e classificar palavras-chave, identificadores, literais e operadores.

10. Análise Sintática

A análise sintática verifica se a sequência de tokens gerada pela análise léxica está estruturada de acordo com as regras da gramática da linguagem. Ela constrói uma árvore de sintaxe abstrata (AST), que representa a estrutura hierárquica do programa e suas expressões. A análise sintática é essencial para garantir que o código-fonte seja válido.

11. Análise Semântica

A análise semântica vai além da verificação da sintaxe, garantindo que o programa esteja de acordo com as regras semânticas da linguagem, como tipos de dados, escopo de variáveis e uso correto de funções. Essa fase é responsável por detectar erros de tipo e inconsistências na execução do programa.

12. Tradução Dirigida por Sintaxe

A tradução dirigida por sintaxe envolve a tradução do código-fonte em uma representação intermediária com base na estrutura sintática do programa. Utiliza-se as regras da gramática para guiar a tradução de expressões e instruções em uma linguagem intermediária que será posteriormente otimizada e convertida em código de máquina.

13. Geração de Código Intermediário

A geração de código intermediário é a fase onde o compilador converte o código fonte em uma representação intermediária (IR). Esse código intermediário é mais fácil de otimizar e pode ser alvo de

transformações para gerar código final eficiente. Além disso, ele é independente da plataforma, facilitando a portabilidade.

14. Otimizador de Código

O otimizador de código é responsável por melhorar o código intermediário, reduzindo seu tamanho e melhorando seu desempenho. Ele pode realizar várias transformações, como eliminação de código redundante, simplificação de expressões e otimização de loops, com o objetivo de gerar um código final mais rápido e eficiente.

15. Gerador de Código

O gerador de código é a fase final do compilador, onde o código intermediário otimizado é convertido em código de máquina ou código para uma linguagem de baixo nível. O gerador de código precisa ser adaptado à arquitetura do processador e à plataforma alvo para garantir a execução correta e eficiente do programa.
