Trabalho de Compiladores – Projeto de um Mini-Compilador

Desenvolver um mini-compilador para uma linguagem de programação de pequeno porte (definida pelo grupo), abrangendo desde a análise léxica até a geração de código otimizado. O projeto deve permitir consolidar todos os conceitos estudados ao longo da disciplina. O conteúdo aborda Introdução a Compiladores; Revisão de Linguagens Formais e Autômatos, e Paradigmas de Programação Análise Léxica Análise Sintática Tradução dirigida por Sintaxe Geração de Código Intermediário Ambientes de Execução Geração de Código Otimizações Independente de Máquina Paralelismo de Instrução Otimização de paralelismo e localidade Aplicações relacionadas à teoria de compiladores.

Estrutura do Trabalho

1. Introdução ao Projeto

- Definição de uma linguagem simplificada (ex.: MiniLang).
- Especificação de elementos básicos: variáveis, expressões aritméticas, estruturas de controle (if, while, for), entrada/saída.
- Breve documento descrevendo a linguagem.

2. Revisão de Linguagens Formais e Autômatos

- Definir a gramática formal da linguagem (BNF ou EBNF).
- Representar o autômato finito para a análise léxica.
- Descrever o paradigma adotado (imperativo, estruturado, etc.).

3. Análise Léxica

- Implementar um analisador léxico que identifique tokens: identificadores, números, operadores, palavras-chave.
- Utilizar expressões regulares ou geradores de analisadores (como *Flex*, *JLex* ou código em Python/C++/Java).

4. Análise Sintática

- Implementar um parser para validar a estrutura sintática.
- Utilizar técnicas de análise descendente ou ascendente (LL(1), LR(1)).
- Representar a árvore sintática abstrata (AST).

5. Tradução Dirigida por Sintaxe

- Associar ações semânticas às regras da gramática.
- Gerar a AST anotada com informações de tipos e escopos.

6. Geração de Código Intermediário

- Transformar a AST em código intermediário (ex.: *three-address code*, *quadruples* ou representação em grafo).
- Exemplo: t1 = a + b, t2 = t1 * c.

7. Ambientes de Execução

- Especificar como será o gerenciamento de memória e variáveis.
- Implementar uma tabela de símbolos para armazenar identificadores, tipos e endereços.

8. Geração de Código

- Traduzir o código intermediário para linguagem de máquina simplificada ou bytecode.
- Alternativa: gerar código em C/Python a partir da linguagem fonte.

9. Otimizações Independentes de Máquina

- Aplicar otimizações clássicas: eliminação de código morto, propagação de constantes, simplificação algébrica.

10. Paralelismo de Instrução

- Analisar trechos de código para identificar instruções que podem ser executadas em paralelo.
- Criar uma representação que evidencie dependências entre instruções.

11. Otimização de Paralelismo e Localidade

- Aplicar técnicas de reordenação de laços ou melhoria de acesso à memória.
- Relatar ganhos teóricos de desempenho.

12. Aplicações Relacionadas

- Discutir onde compiladores são aplicados atualmente:
    - Compiladores JIT (Java, .NET, Python).
    - Compiladores para GPU.
    - Aplicações em otimização de código para sistemas embarcados e IoT.
- Relacionar com a experiência prática do projeto.

Cronograma de Desenvolvimento – Trabalho de Compiladores

Semana 1 – Definição do Projeto e Linguagem

- Formação dos grupos (4 alunos).
- Escolha/definição da linguagem simplificada a ser compilada.
- Entrega parcial: Descrição da linguagem e exemplos de programas.

Semana 2 – Gramática e Autômatos

- Definição formal da gramática (BNF ou EBNF).
- Construção do autômato léxico (tokens) e escolha de paradigma de programação (imperativo, estruturado, etc.).
- Entrega parcial: Gramática formal + especificação dos tokens.

Semana 3 – Análise Léxica

- Implementação do analisador léxico (com gerador ou manualmente).
- Testes de reconhecimento de tokens em programas simples.
- Entrega parcial: Código do lexer + relatório de testes.

Semana 4 – Análise Sintática

- Implementação do parser (LL(1), LR(1), ou outra técnica).
- Construção da árvore sintática abstrata (AST).
- Entrega parcial: Código do parser + exemplos de AST.

Semana 5 – Tradução Dirigida por Sintaxe

- Inserção de ações semânticas.
- Criação e uso de tabela de símbolos (variáveis, tipos, escopos).
- Entrega parcial: AST anotada + tabela de símbolos.

Semana 6 – Código Intermediário

- Transformação da AST em código intermediário (ex.: três endereços).
- Representação dos trechos com variáveis temporárias.
- Entrega parcial: Código intermediário de exemplos de programas.

Semana 7 – Ambientes de Execução

- Definição da estrutura de memória/variáveis em tempo de execução.
- Ajustes no compilador para considerar alocação de variáveis.
- Entrega parcial: Descrição/documento sobre ambiente de execução.

Semana 8 – Geração de Código

- Conversão para código final (linguagem de máquina simulada ou *bytecode*).

- Execução de programas completos.
- Entrega parcial: Código final gerado + exemplos executáveis.

Semana 9 – Otimizações

- Aplicar otimizações independentes de máquina: eliminação de código morto, propagação de constantes.
- Analisar paralelismo de instruções e localidade de memória.
- Entrega parcial: Relatório de otimizações aplicadas + exemplos.

Semana 10 – Aplicações e Conclusão

- Relacionar o projeto com aplicações reais de compiladores (JIT, GPUs, sistemas embarcados, etc.).
- Finalizar relatório.
- Preparar apresentação final com demonstração do compilador.
- Entrega final: Relatório completo + código-fonte + apresentação em grupo.