UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Compilador Parcial em C

Felipe Henrique Verones Pereria dos Santos

Disciplina: Construção de Compiladores
Professor: Alison Roberto Panisson

Araranguá
4 de julho de 2023

Conteúdo

1 Resumo

Este relatório descreve a implementação de um reconhecedor de estruturas em linguagem C utilizando a biblioteca PLY (Python Lex-Yacc). A biblioteca PLY é uma implementação puramente Python do popular conjunto de ferramentas Lex e Yacc, comumente usadas para criar analisadores léxicos e sintáticos. A implementação apresentada neste relatório é capaz de reconhecer estruturas de controle de fluxo, declarações de variáveis, expressões aritméticas, lógicas e de comparação.

2 Introdução

A construção de compiladores é uma disciplina fundamental na ciência da computação, que envolve a tradução de código-fonte escrito em uma linguagem de programação para uma forma que possa ser executada por uma máquina. Esta tarefa é geralmente dividida em várias etapas, incluindo análise léxica, análise sintática, análise semântica, otimização de código e geração de código.

A análise léxica e sintática, as etapas focadas neste relatório, são responsáveis por transformar o código-fonte em uma representação estruturada que pode ser manipulada mais facilmente nas etapas subsequentes do processo de compilação. A análise léxica divide o código-fonte em tokens, que são as unidades básicas de significado na linguagem. A análise sintática, por sua vez, organiza esses tokens em uma estrutura de árvore que reflete a gramática da linguagem.

A biblioteca PLY, desenvolvida por David Beazley, é uma implementação Lex e Yacc em Python que fornece uma interface para a construção de analisadores léxicos e sintáticos. A PLY é altamente flexível e pode ser usada para analisar quase qualquer linguagem de programação.

3 Teoria de Compiladores

A construção de um compilador envolve uma série de fases que transformam o código fonte em um programa executável. Essas fases incluem análise léxica, análise sintática, análise semântica, otimização de código e geração de código. Cada fase tem uma função específica e contribui para a correta tradução do código fonte.

3.1 Algoritmos de Construção de Compiladores

A construção de um compilador envolve uma série de algoritmos complexos que são usados para analisar o código fonte e gerar o código de máquina. Esses algoritmos são geralmente divididos em duas fases principais: a análise (ou front-end) e a síntese (ou back-end).

3.2 Análise Léxica

A análise léxica é a primeira fase do processo de compilação e é responsável por converter o código fonte em uma sequência de tokens. Cada token representa um componente lógico do código, como uma palavra reservada, um identificador, um número ou um operador. Esta fase é crucial para simplificar o processo de análise sintática, pois reduz a complexidade do código fonte ao identificar e agrupar caracteres que formam unidades lógicas. A análise léxica é geralmente implementada usando um autômato finito determinístico (DFA).

3.3 Análise Sintática

A análise sintática é a segunda fase do processo de compilação e é responsável por verificar se a sequência de tokens produzida pela análise léxica segue as regras gramaticais da linguagem. Nessa fase, o analisador pega a sequência de tokens gerada pela análise léxica e tenta determinar sua estrutura. Isso é feito construindo uma árvore de sintaxe abstrata (AST) que representa a estrutura lógica do código. Esta fase é geralmente implementada usando um dos muitos algoritmos de análise de gramática, como o LL, o LR, o LALR, etc. Se a sequência de tokens é válida, a análise sintática produz a árvore de sintaxe abstrata que representa a estrutura lógica do código. Esta fase é fundamental para entender a organização e a estrutura do código fonte.

3.4 Análise Semântica

A análise semântica é a terceira fase do processo de compilação e é responsável por verificar se o código faz sentido do ponto de vista semântico. Isso inclui verificar se as variáveis foram declaradas antes do uso, se os tipos de dados são compatíveis nas operações e se as funções são chamadas com o número correto de argumentos. Esta fase é essencial para garantir que o código fonte seja logicamente correto e possa ser traduzido em um programa executável.

A análise semântica é uma das partes mais complexas da construção de um compilador. Ela é responsável por verificar se o programa faz sentido do ponto de vista

lógico. Isso inclui a verificação de tipos, a verificação de escopo, a verificação de funções e procedimentos, entre outros.

Na implementação, a análise semântica é realizada durante a análise sintática. Quando a árvore de sintaxe abstrata é construída, realizamos várias verificações semânticas. Por exemplo, quando um nó que representa uma declaração de variável é criado, verificamos se a variável já foi declarada no mesmo escopo. Se a variável já foi declarada, emitimos um erro.

A verificação de tipos é realizada quando um nó que representa uma operação é criado. Nós verificamos se os tipos dos operandos são compatíveis com a operação. Por exemplo, se a operação é uma adição, verifica-se se os operandos são de um tipo numérico. Se os operandos não são de um tipo numérico, emitimos um erro.

A verificação de funções e procedimentos é realizada quando um nó que representa uma chamada de função ou procedimento é criado. Nós verificamos se a função ou procedimento foi declarado e se o número e tipos de argumentos são compatíveis com a declaração. Se a função ou procedimento não foi declarado ou se os argumentos não são compatíveis, nós emitimos um erro.

3.5 Geração de Código

A geração de código é a última fase de um compilador. Seu objetivo é pegar a AST gerada pela análise sintática e gerar o código de máquina correspondente. A geração de código é geralmente implementada como outra caminhada na AST.

4 PLY e Algoritmos de Compilação

A Python Lex-Yacc (PLY) é uma biblioteca Python que implementa as funcionalidades das ferramentas de análise léxica e sintática Lex e Yacc, amplamente utilizadas na construção de compiladores. A PLY permite definir a gramática da linguagem de forma declarativa, usando regras de produção escritas como funções Python. Além disso, a PLY suporta ações semânticas, que são blocos de código Python que são executados quando uma regra de produção é aplicada.

PLY usa DFAs para a análise léxica. Você define um conjunto de expressões regulares e funções associadas, e PLY gera um DFA que pode analisar o código fonte e gerar a sequência de tokens.

A PLY utiliza o algoritmo LALR(1) (Look-Ahead Left-to-right Rightmost derivation) para análise sintática, que é uma variação do algoritmo LR(1) que consome menos memória. Este algoritmo é capaz de analisar a maioria das gramáticas livre de contexto, tornando-o adequado para a implementação de analisadores para muitas linguagens de programação. Basta você definir um conjunto de regras de produção e funções associadas, e PLY gera um analisador LALR(1) que pode construir a AST a partir da sequência de tokens.

A análise semântica e a geração de código não são implementadas diretamente por PLY. Em vez disso, você implementa essas fases como funções associadas às regras de produção. Quando uma regra de produção é aplicada durante a análise sintática, a função associada é chamada, permitindo que você realize a análise semântica e a geração de código.

4.1 Implementação

A implementação consiste em duas partes principais: o analisador léxico e o analisador sintático. O analisador léxico foi implementado usando a função lex.lex(), que recebe como entrada uma lista de tokens e uma série de funções que definem como esses tokens são reconhecidos no texto de entrada. Os tokens definidos incluem palavras reservadas da linguagem C, operadores aritméticos e de comparação, delimitadores e identificadores.

O analisador sintático foi implementado usando a função yacc.yacc(), que recebe como entrada uma série de regras de produção. Cada regra de produção é definida por uma função Python, cujo docstring define a regra em termos de tokens e outras regras de produção. A função em si implementa a ação semântica associada à regra, que é executada quando a regra é aplicada durante a análise.

4.2 Ações semânticas Implementadas

No código desenvolvido, foi implementado um analisador léxico e sintático para estruturas simples da linguagem C usando a biblioteca PLY.

O analisador léxico implementado é capaz de reconhecer uma variedade de tokens, incluindo palavras reservadas, identificadores, operadores, números e strings. Ainda, foi implementada a funcionalidade de ignorar comentários e espaços em branco.

O analisador sintático, por outro lado, é capaz de analisar uma variedade de estruturas de controle de fluxo, incluindo instruções if, while e for. Também pode analisar declarações de variáveis, atribuições e expressões aritméticas e lógicas.

As ações semânticas implementadas no código são principalmente para manipulação da tabela de símbolos. A tabela de símbolos é usada para armazenar informações sobre identificadores (como variáveis e funções) que são encontrados durante a análise.

As ações semânticas implementadas incluem:

- **Adicionar variáveis à tabela de símbolos:** Quando uma declaração de variável é encontrada, a variável é adicionada à tabela de símbolos com seu tipo, nome e, se houver, também com seu valor inicial.
- **Verificação de declaração de variáveis:** Antes de uma variável ser usada em uma expressão ou atribuição, verifica-se se ela foi declarada, ou seja, se ela existe na tabela de símbolos. Se ela já foi declarada em outra parte do código, um erro é gerado.
- **Atribuição e atualização de valores de variáveis:** Quando uma atribuição adequada é encontrada, o valor da variável é atualizado na tabela de símbolos.
- **Verificação de compatibilidade de tipos:** Durante uma atribuição, o analisador verifica se o tipo do valor a ser atribuído é compatível com o tipo da variável. Se os tipos forem incompatíveis, um erro é gerado.
- **Verificação de expressões nulas:** O analisador verifica se as expressões usadas em operações aritméticas e atribuições não são nulas. Se uma expressão é nula, um erro é gerado.

- **Operações Aritméticas:** O analisador realiza operações aritméticas simples como incremento, decremento, adição, subtração, divisão, multiplicação, módulo e potência, verificando os tipos. Caso uma das expressões da operação seja nula ou mal formulada, um erro é gerado. O mesmo ocorre se uma operação não for suportada para expressões não identificadoras, por exemplo, como no caso das operações de incremento (++) e decremento (–).

Essas ações semânticas ajudam a garantir que o programa C que está sendo analisado é semanticamente correto. Se um erro semântico é encontrado (como o uso de uma variável não declarada), uma exceção é lançada e a análise é interrompida.

Em resumo, o código que você implementou é uma parte fundamental de um compilador para a linguagem C. Ele é capaz de analisar a sintaxe de um programa C e verificar sua correção semântica. No entanto, ainda faltam algumas partes para um compilador completo, como a geração de código intermediário, a otimização de código e a geração de código de máquina.

5 Conclusão

A implementação de um analisador léxico e sintático com PLY é uma tarefa complexa que requer um entendimento profundo tanto da linguagem de programação alvo quanto dos princípios de análise léxica e sintática. No entanto, a flexibilidade e a facilidade de uso da PLY tornam essa tarefa mais gerenciável.

A implementação aqui descrita é capaz de reconhecer e analisar uma subconjunto significativo da linguagem C. As ações semânticas implementadas permitem a construção de uma tabela de símbolos e a verificação de tipos, que são etapas fundamentais na compilação de um programa.

Este trabalho representa um passo importante na direção de um compilador completo para a linguagem C implementado em Python. Trabalhos futuros poderiam expandir esta implementação para cobrir mais recursos da linguagem e implementar etapas adicionais do processo de compilação, como a otimização de código e a geração de código de máquina.

A implementação apresentada neste relatório demonstra o poder e a flexibilidade da PLY como uma ferramenta para a construção de analisadores léxicos e sintáticos. Espera-se que este trabalho possa servir como um recurso útil para outros desenvolvedores interessados em implementar seus próprios analisadores usando a PLY.