Construção de um Analisador Sintático

Disciplina: Construção de Compiladores (2025.1)
Discentes: Kailane Lisley de Araújo Silva e Gabrielly Gouveia da Silva Feitosa

1. Objetivo

Construir um analisador sintático capaz de verificar a correção gramatical de sequências de tokens geradas para uma linguagem fictícia, utilizando como base as expressões regulares e a Gramática Livre de Contexto (GLC) fornecida.

2. Objetivos Específicos

2.1. Análise e compreensão das especificações da linguagem

2.1.1. Linguagem Fictícia

A linguagem de programação fictícia utilizada para realizar a análise sintática possui características comuns a linguagens como C ou Java. Tal linguagem suporta:

- Declaração de variáveis de tipos básicos (INT, FLOAT, CHAR, BOOLEAN).
- Declaração de funções (incluindo VOID).
- Atribuições.
- Estruturas de controle condicionais (IF-ELSE) e de repetição (WHILE).
- Expressões aritméticas e de comparação.
- Chamadas de função com argumentos.
- Blocos de código delimitados por chaves.
- Comando de retorno (RETURN).

2.1.2. Sequência de Tokens

O analisador léxico (Não implementado neste projeto) como sendo a primeira etapa de um processo de compilação gera uma sequência de tokens que será passada para o analisador sintático. Seguindo a estrutura NOME_DO_TOKENS lexema CATEGORIA_DO_TOKEN temos a seguinte sequência de tokens hipotética gerada com base nas expressões regulares e GLC fornecidas a fim de ser utilizada como base para o desenvolvimento esse projeto:

INT int PALAVRA_RESERVADA
FLOAT float PALAVRA_RESERVADA
CHAR char PALAVRA_RESERVADA
BOOLEAN boolean PALAVRA_RESERVADA
VOID void PALAVRA_RESERVADA
IF if PALAVRA_RESERVADA
ELSE else PALAVRA_RESERVADA
FOR for PALAVRA_RESERVADA
WHILE while PALAVRA_RESERVADA
SCANF scanf PALAVRA_RESERVADA

PRINTF printf PALAVRA_RESERVADA
MAIN main PALAVRA_RESERVADA
RETURN return PALAVRA_RESERVADA
NUM_INT 42 NUMERO_INTEIRO
NUM_DEC 3.14 NUMERO_DECIMAL
ID contador IDENTIFICADOR
ID soma IDENTIFICADOR
ID resultado IDENTIFICADOR
TEXTO "Hello, World!" CONSTANTE_TEXTO
== == OPERADOR_COMPARACAO
!= != OPERADOR_COMPARACAO
<= <= OPERADOR_COMPARACAO
>= >= OPERADOR_COMPARACAO
+ + OPERADOR_ARITMETICO
- - OPERADOR_ARITMETICO
* * OPERADOR_ARITMETICO
/ / OPERADOR_ARITMETICO
% % OPERADOR_ARITMETICO
= = OPERADOR_ATRIBUICAO
&& && OPERADOR_LOGICO
|| || OPERADOR_LOGICO
! ! OPERADOR_LOGICO
((SIMBOLO_ESPECIAL
)) SIMBOLO_ESPECIAL
{ { SIMBOLO_ESPECIAL
} } SIMBOLO_ESPECIAL
[[SIMBOLO_ESPECIAL
]] SIMBOLO_ESPECIAL
; ; SIMBOLO_ESPECIAL
, , SIMBOLO_ESPECIAL
COMMENT // COMENTARIO

2.1.3. Gramática Livre de Contexto (GLC) para linguagem fictícia

Após análise das Expressões Regulares e da GLC fornecidas para o desenvolvimento desse projeto, utilizamos como base para a implementação a seguinte versão simplificada da gramática original, na qual foi ajustada para ser LL(1) e facilitar a análise preditiva:

I) Terminais

1) INT
2) FLOAT
3) CHAR
4) BOOLEAN
5) VOID

6) ID
7) NUM
8) IF
9) ELSE
10) WHILE
11) RETURN
12) COMP (Operadores de Comparação)
13) OP_ARIT (Operadores Aritméticos)
14) ABRE_PAREN
15) FECHA_PAREN
16) ABRE_CHAVE
17) FECHA_CHAVE
18) PONTO_VIRGULA
19) VIRGULA, ATRIBUICAO
20) T_EOF (Terminal Final)
21) NUM_TERMINAIS (Contador dos Terminais)

II) Não Terminais

1) Programa
2) Declaracoes
3) Declaracao
4) Tipo
5) DeclaracaoResto
6) Parametros
7) ListaParametros
8) RestoListaParametros
9) Bloco
10) Comandos
11) Comando
12) ComandoCondicional
13) ElseOpcional
14) ExpressaoOpcional
15) Expressao
16) ExpressaoResto
17) Termo
18) TermoResto
19) Fator
20) FatorResto
21) Argumentos
22) ListaArgumentos
23) RestoListaArgumentos
24) NUM_NAO_TERMINAIS (Contador dos Não Terminais)

III) Produções

0) Programa' -> Programa T_EOF
1) Programa -> Declaracoes
2) Declaracoes -> Declaracao Declaracoes
3) Declaracoes -> epsilon
4) Declaracao -> Tipo ID DeclaracaoResto
5) Tipo -> INT
6) Tipo -> FLOAT
7) Tipo -> CHAR
8) Tipo -> BOOLEAN
9) Tipo -> VOID
10) DeclaracaoResto -> PONTO_VIRGULA
11) DeclaracaoResto -> ATRIBUICAO Expressao PONTO_VIRGULA
12) DeclaracaoResto -> ABRE_PAREN Parametros FECHA_PAREN Bloco
13) Parametros -> ListaParametros
14) Parametros -> epsilonListaParametros -> Tipo ID RestoListaParametros
15) RestoListaParametros -> VIRGULA Tipo ID RestoListaParametros
16) RestoListaParametros -> epsilon
17) Bloco -> ABRE_CHAVE Comandos FECHA_CHAVE
18) Comandos -> Comando Comandos
19) Comandos -> epsilon
20) Comando -> Declaracao
21) Comando -> ID ATRIBUICAO Expressao PONTO_VIRGULA
22) Comando -> ComandoCondicional
23) Comando -> Bloco
24) Comando -> RETURN ExpressaoOpcional PONTO_VIRGULA
25) ComandoCondicional -> IF ABRE_PAREN Expressao
26) FECHA_PARE Comando ElseOpcional
27) ComandoCondicional -> WHILE ABRE_PAREN Expressao
28) FECHA_PAREN Comando
29) ElseOpcional -> ELSE Comando
30) ElseOpcional -> epsilon
31) ExpressaoOpcional -> Expressao
32) ExpressaoOpcional -> epsilon
33) Expressao -> Termo ExpressaoResto
34) ExpressaoResto -> COMP Termo ExpressaoResto
35) ExpressaoResto -> epsilon
36) Termo -> Fator TermoResto
37) TermoResto -> OP_ARIT Fator TermoResto
38) TermoResto -> epsilon
39) Fator -> ID FatorResto
40) Fator -> NUM
41) Fator -> ABRE_PAREN Expressao FECHA_PAREN

42) FatorResto -> ABRE_PAREN Argumentos FECHA_PAREN
43) FatorResto -> epsilon
44) Argumentos -> ListaArgumentos
45) Argumentos -> epsilon
46) ListaArgumentos -> Expressao RestoListaArgumentos
47) RestoListaArgumentos -> VIRGULA Expressao RestoListaArgumentos
48) RestoListaArgumentos -> epsilon

2.2. Abordagem de Análise Sintática

Foi escolhida a abordagem de **Análise Sintática Preditiva com Tabela M (LL(1))** implementada manualmente na linguagem C. Ela utiliza uma pilha e uma tabela de análise (Tabela M) para determinar qual produção aplicar com base no não-terminal no topo da pilha e no próximo token de entrada.
Esta escolha foi feita pois permite aplicar diretamente os conceitos teóricos de análise preditiva, cálculo de conjuntos FIRST/FOLLOW e construção de tabela M, além de oferecer controle total sobre o processo de análise e tratamento de erros.

2.2.1. Conjuntos First e Follow

Para construir a Tabela M, foram calculados os conjuntos FIRST e FOLLOW para cada não-terminal da gramática.

2.2.1.1. Conjuntos FIRST

O conjunto FIRST(α) contém os terminais que podem iniciar uma sequência derivada de α, incluindo ε se α pode derivar a cadeia vazia.

FIRST(Programa') = {INT, FLOAT, CHAR, BOOLEAN, VOID, EOF, epsilon}
FIRST(Programa) = {INT, FLOAT, CHAR, BOOLEAN, VOID, epsilon}
FIRST(Declaracoes) = {INT, FLOAT, CHAR, BOOLEAN, VOID, epsilon}
FIRST(Declaracao) = {INT, FLOAT, CHAR, BOOLEAN, VOID}
FIRST(Tipo) = {INT, FLOAT, CHAR, BOOLEAN, VOID}
FIRST(DeclaracaoResto) = {PONTO_VIRGULA, ATRIBUICAO, ABRE_PAREN}
FIRST(Parametros) = {INT, FLOAT, CHAR, BOOLEAN, VOID, epsilon}
FIRST(ListaParametros) = {INT, FLOAT, CHAR, BOOLEAN, VOID}
FIRST(RestoListaParametros) = {VIRGULA, epsilon}
FIRST(Bloco) = {ABRE_CHAVE}
FIRST(Comandos) = {INT, FLOAT, CHAR, BOOLEAN, VOID, ID, IF, WHILE, ABRE_CHAVE, RETURN, epsilon}
FIRST(Comando) = {INT, FLOAT, CHAR, BOOLEAN, VOID, ID, IF, WHILE, ABRE_CHAVE, RETURN}
FIRST(ComandoCondicional) = {IF, WHILE}
FIRST(ElseOpcional) = {ELSE, epsilon}
FIRST(ExpressaoOpcional) = {ID, NUM, ABRE_PAREN, epsilon}
FIRST(Expressao) = {ID, NUM, ABRE_PAREN}

FIRST(ExpressaoResto) = {COMP, epsilon}
FIRST(Termo) = {ID, NUM, ABRE_PAREN}
FIRST(TermoResto) = {OP_ARIT, epsilon}
FIRST(Fator) = {ID, NUM, ABRE_PAREN}
FIRST(FatorResto) = {ABRE_PAREN, epsilon}
FIRST(Argumentos) = {ID, NUM, ABRE_PAREN, epsilon}
FIRST(ListaArgumentos) = {ID, NUM, ABRE_PAREN}
FIRST(RestoListaArgumentos) = {VIRGULA, epsilon}

2.2.1.2. Conjuntos FOLLOW

O conjunto FOLLOW(A) contém os terminais que podem aparecer imediatamente após o não-terminal A em alguma forma de sentença válida, incluindo o marcador de fim de entrada T_EOF ($) se A pode ser o último símbolo.

FOLLOW(Programa') = {}
FOLLOW(Programa) = {T_EOF}
FOLLOW(Declaracoes) = {T_EOF}
FOLLOW(Declaracao) = {INT, FLOAT, CHAR, BOOLEAN, VOID, T_EOF}
FOLLOW(Tipo) = {ID}
FOLLOW(DeclaracaoResto) = FOLLOW(Declaracao) = {INT, FLOAT, CHAR, BOOLEAN, VOID, T_EOF, ID, IF, WHILE, ABRE_CHAVE, RETURN, FECHA_CHAVE}
FOLLOW(Parametros) = {FECHA_PAREN}
FOLLOW(ListaParametros) = {FECHA_PAREN}
FOLLOW(RestoListaParametros) = {FECHA_PAREN}
FOLLOW(Bloco) = FOLLOW(DeclaracaoResto) + FOLLOW(Comando) = {INT, FLOAT, CHAR, BOOLEAN, VOID, T_EOF, ID, IF, WHILE, ABRE_CHAVE, RETURN, FECHA_CHAVE, ELSE}
FOLLOW(Comandos) = {FECHA_CHAVE}
FOLLOW(Comando) = FOLLOW(Comandos) + FOLLOW(ElseOpcional) = { FECHA_CHAVE, INT, FLOAT, CHAR, BOOLEAN, VOID, ID, IF, WHILE, ABRE_CHAVE, RETURN, T_EOF}
FOLLOW(ComandoCondicional) = FOLLOW(Comando)
FOLLOW(ElseOpcional) = FOLLOW(Comando)
FOLLOW(ExpressaoOpcional) = {PONTO_VIRGULA}
FOLLOW(Expressao) = {PONTO_VIRGULA, FECHA_PAREN, VIRGULA}
FOLLOW(ExpressaoResto) = FOLLOW(Expressao)
FOLLOW(Termo) = FOLLOW(ExpressaoResto) + FIRST(ExpressaoResto) - {epsilon} = {PONTO_VIRGULA, FECHA_PAREN, VIRGULA, COMP}
FOLLOW(TermoResto) = FOLLOW(Termo)
FOLLOW(Fator) = FOLLOW(TermoResto) + FIRST(TermoResto) - {epsilon} = {PONTO_VIRGULA, FECHA_PAREN, VIRGULA, COMP, OP_ARIT}
FOLLOW(FatorResto) = FOLLOW(Fator)
FOLLOW(Argumentos) = {FECHA_PAREN}
FOLLOW(ListaArgumentos) = {FECHA_PAREN}
FOLLOW(RestoListaArgumentos) = {FECHA_PAREN}

2.2.2. Tabela M

A Tabela M foi construída usando as seguintes regras:

Para cada produção A -> α:

Para cada terminal a em FIRST(α), adicione A -> α em M[A, a].
Se ε está em FIRST(α):

- Para cada terminal b em FOLLOW(A), adicione A -> α (que é A -> ε) em M[A, b].
- Se T_EOF ($) está em FOLLOW(A), adicione A -> ε em M[A, T_EOF].

A tabela completa e preenchida se encontra em anexo a esta documentação.

2.3. Implementação

Para implementar o analisador sintático manualmente foi utilizada a linguagem de programação C e possui as seguintes estruturas de dados principais:

- **Enums**: Foram definidos enum para representar os Não-Terminais e os Terminais, garantindo consistência com a gramática utilizada na implementação.
- **Pilha**: Uma pilha simples implementada com um array (int pilha[MAX_PILHA]) e um índice de topo (topo) é usada para armazenar os símbolos gramaticais durante a análise.
- **Tabela M**: Uma matriz bidimensional int tabelaM[NUM_NAO_TERMINAIS] [NUM_TERMINAIS] armazena os identificadores das produções a serem aplicadas, indexada pelos enums de não-terminais e terminais.
- **Tokens**: A entrada do analisador é um array de estruturas Token (ou similar), onde cada estrutura contém pelo menos o tipo do token (o enum Terminal).
- **Tabela M:** Inicializa a Tabela M e preenche com as produções.
- **Leitura de Tokens**: Lê a sequência de tokens do arquivo fornecido como argumento.
- **Mapeamento**: Converte os nomes dos tokens lidos para os enum Terminal correspondentes usando mapearToken.
- **Funções da Pilha**: empilhar, desempilhar, verTopo, pilhaVazia gerenciam a pilha de análise.
- **aplicarProducao**: Contém a lógica para empilhar os símbolos corretos para cada produção da gramática.
- **Main**: Implementa o algoritmo de parsing LL(1), comparando o topo da pilha com o token atual e consultando a Tabela M.

O código completo implementado se encontra em anexo a esta documentação.

2.4. Testes e Validação

Foram criados 10 casos de teste para validar o bom funcionamento do analisador sintático considerando sequências sintaticamente corretas (declarações simples, condicionais, laços, etc.) e sequências com erros sintáticos comuns (ponto e vírgula faltando, parênteses desbalanceados, ordem incorreta de comandos, etc.)

2.4.1. Casos de Teste

Os arquivos Tokens1.txt a Tokens10.txt servem como casos de teste:

Caso 1: Sequência correta simples (declaração única):

INT int PALAVRA_RESERVADA
ID a IDENTIFICADOR
; ; SIMBOLO_ESPECIAL

Cadeia: int a;

Saída Espera: Análise sintática concluída com sucesso!

Caso 2: Sequência correta composta (múltiplas declarações):

INT int PALAVRA_RESERVADA
ID a IDENTIFICADOR
; ; SIMBOLO_ESPECIAL
FLOAT float PALAVRA_RESERVADA
ID b IDENTIFICADOR
; ; SIMBOLO_ESPECIAL
FLOAT float PALAVRA_RESERVADA
ID c IDENTIFICADOR
; ; SIMBOLO_ESPECIAL

Cadeia: int a;
float b;
float c;

Saída Espera: Análise sintática concluída com sucesso!

Caso 3: Sequência incorreta - falta do ponto e vírgula:

INT int PALAVRA_RESERVADA
ID b IDENTIFICADOR

Cadeia: int b

Saída Espera: Erro sintático: esperado PONTO_VIRGULA, encontrado EOF (fim de tokens).

Caso 4: Sequência incorreta - token inesperado:

INT int PALAVRA_RESERVADA
; ; SIMBOLO_ESPECIAL
Cadeia: int;

Saída Esperada: Erro sintático: esperado ID, encontrado ; .

Caso 5: Sequência incorreta - token desconhecido:

INT int PALAVRA_RESERVADA
ID c IDENTIFICADOR
DESCONHECIDO ??? ERROR
; ; SIMBOLO_ESPECIAL

Cadeia: int c ???;

Saída Esperada: Erro sintático: nenhuma produção para o não-terminal atual com token DESCONHECIDO.

Caso 6: Sequência incompleta - tipo sem ID e ponto e vírgula:

FLOAT float PALAVRA_RESERVADA

Cadeia: float

Saída Esperada: Erro sintático: esperado ID, mas encontrou EOF.

Caso 7: Sequência incorreta - ordem trocada:

ID x IDENTIFICADOR
INT int PALAVRA_RESERVADA
; ; SIMBOLO_ESPECIAL

Cadeia: x int;

Saída Esperada: Erro sintático: nenhuma produção para PROGRAMA com token ID.

Caso 8: Sequência correta com diferentes tipos:

BOOLEAN boolean PALAVRA_RESERVADA
ID abc IDENTIFICADOR
; ; SIMBOLO_ESPECIAL
CHAR char PALAVRA_RESERVADA
ID letra IDENTIFICADOR
; ; SIMBOLO_ESPECIAL

Cadeia: boolean abc;
char letra;

Saída Espera: Análise sintática concluída com sucesso!

Caso 9: Sequência correta com VOID:

VOID void PALAVRA_RESERVADA
ID func IDENTIFICADOR

; ; SIMBOLO_ESPECIAL

Cadeia: void func;

Saída Espera: Análise sintática concluída com sucesso!

Caso 10: Sequência incorreta - apenas um ponto e vírgula:
; ; SIMBOLO_ESPECIAL

Cadeia: ;

Saída Esperada: Erro sintático: nenhuma produção para PROGRAMA com token PONTO_VIRGULA.

2.4.2. Como Executar os Testes

1) **Compilar**: gcc analisadorSin.c -o analisadorSin -Wall
2) **Executar**: ./analisadorSin Tokens*.txt (substitua * pelo número do caso de teste desejado).
3) **Observar a Saída**: Verificar se o analisador reporta "sucesso" para entradas corretas e "erro" para entradas incorretas.

3. Conclusão

3.1. Resultados

O projeto implementou com sucesso um analisador sintático preditivo LL(1) em C para uma linguagem de programação fictícia. Foram aplicados os conceitos teóricos de GLC, conjuntos FIRST e FOLLOW, e Tabela M.

3.2. Desafios e Soluções

- **Inconsistências:** O principal desafio foi alinhar a documentação (GLC, Conjuntos First e Follow, Tabela M e etc) com a implementação prática. A solução foi padronizar a gramática, os conjuntos e a tabela M.
- **Complexidade da Gramática:** A gramática original era complexa e difícil de trabalhar. A solução foi realizar a simplificação e ajustes para transforma-la em LL(1) para que a analise preditiva fosse realizada.
- **Implementação da Tabela M**: Foi necessária muta atenção ao preencher manualmente a tabela M em C pois é um trabalho que está muito propenso a ter erros.

3.3. Arquivos do Projeto

O projeto completo com todos os arquivos está no GitHub no seguinte repositório: https://github.com/KailaneLisley/Analisador_Sinatico