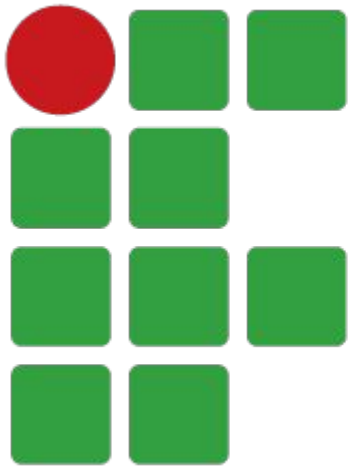


INSTITUTO FEDERAL
Minas Gerais
Campus Bambuí

Compiladores

Professor: Cláudio Ribeiro de Sousa
Feito por: Julia Gabriella, Johnattan Silva e Henrique Araujo

Linguagem Rubrum

- Destaque pela abordagem única com o uso do latim nas palavras reservadas;
- Experiência diferenciada e um ponto de partida acessível para iniciantes em programação;
- Possui um escopo simples.

Linguagem Rubrum

- Estruturas Básicas da Linguagem:
  - Laço, condicionais, declaração de variáveis;
  - Operadores aritméticos, relacionais e lógicos.
- A seguir, é mostrado a Tabela 1 de tokens da linguagem, que se trata de uma estrutura de dados que armazena informações sobre cada token identificado.

Tabela de Tokens

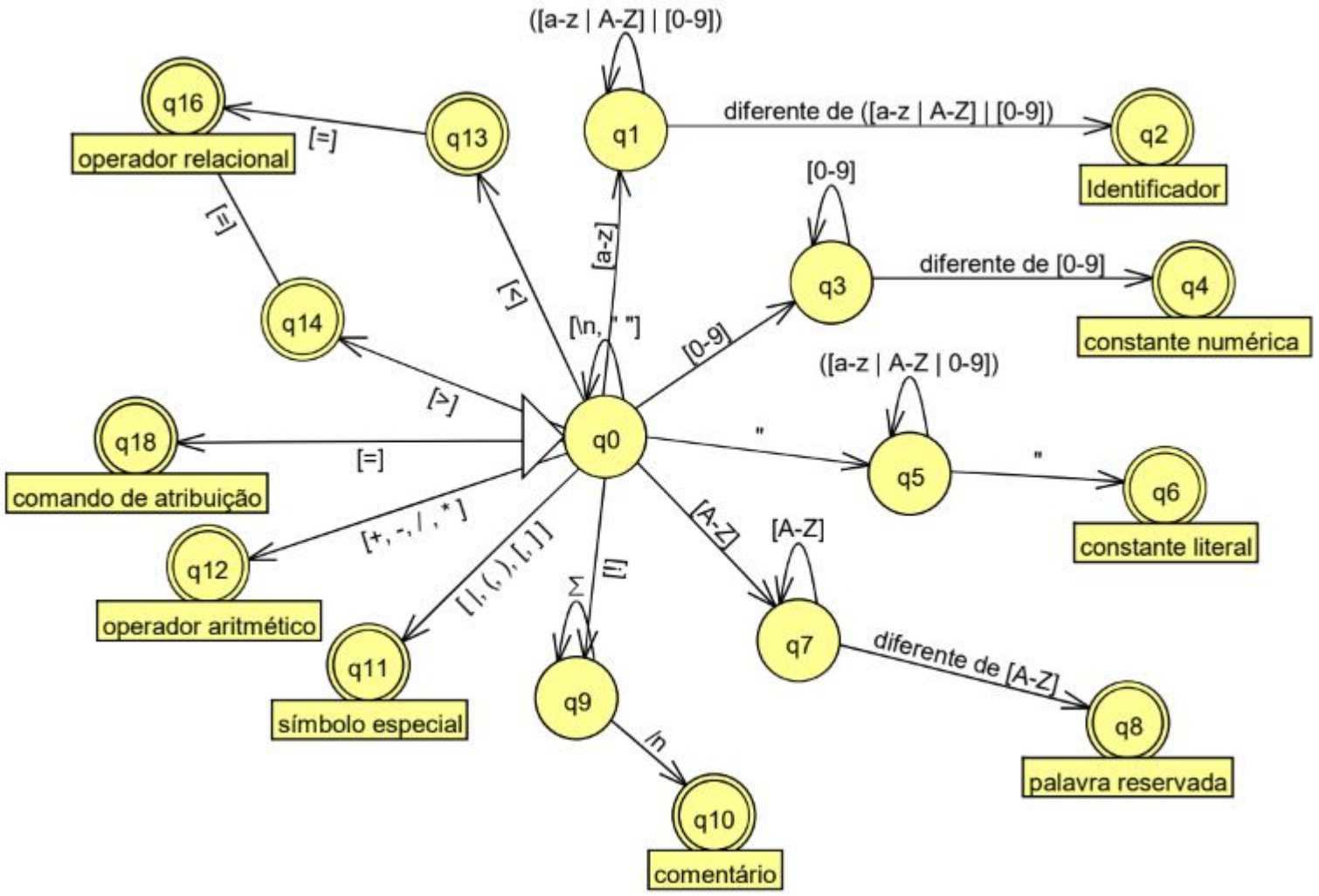
Expressão Regular	Token	Descrição
[0-9]		Dígitos
[a-z | A-Z]		Caracteres
(a-z) (a-z | A-Z | 0-9)*	<id, >	Identificador
(0-9)+	<num, >	Constante Numérica
" (a-z | A-Z | 0-9)* "	<literal, "">	Constante Literal
SATUS	<SATUS, >	Palavra Reservada
LITTERAE	<LITTERAE, >	Palavra Reservada
NUMERUS	<NUMERUS, >	Palavra Reservada
SI	<SI, >	Palavra Reservada
ALIUD	<ALIUD, >	Palavra Reservada
DUM	<DUM, >	Palavra Reservada
LEGERE	<LEGERE, >	Palavra Reservada
SCRIBERE	<SCRIBERE, >	Palavra Reservada
REDITUS	<REDITUS, >	Palavra Reservada
!		Comentário
; = |	<|,>	Símbolo especial
(	<(,>	Símbolo especial
)	<),>	Símbolo especial
[	<[,>	Símbolo especial
]	<],>	Símbolo especial
+	<+,>	Operador Aritmético
-	<-,>	Operador Aritmético
/	</,>	Operador Aritmético
*	<*,>	Operador Aritmético
<	<<,>	Operador Relacional
>	<>,>	Operador Relacional
<=	<<=,>	Operador Relacional
>=	<>=,>	Operador Relacional
=	<=,>	Comando de Atribuição
AUT	<AUT,>	Operador lógico
ET	<ET,>	Operador lógico
NO	<NO,>	Operador lógico
VERUM	<VERUM, >	Booleano
FALSUS	<FALSUS, >	Booleano

Fonte: Elaborado pelos autores, 2022

Autômato

- Autômato finito determinístico (AFD) são os que possuem, para cada combinação de estado e entrada, uma única transição aplicável.
- Desempenha um papel crucial na implementação da análise léxica para identificação de palavras e outros símbolos específicos.

Autômato da Linguagem



Gramática

- A gramática é conjunto formal de regras que define a estrutura sintática de uma linguagem, permitindo que o compilador compreenda e processe corretamente o código-fonte.
- A linguagem Rubrum utiliza a classe de gramática preditiva LL(1), que permite análise utilizando apenas um símbolo de entrada por vez.

Gramática da Linguagem

MAIN	->	SATUS '(' DECLARA BLOCO ')'
DECLARA	->	VAR DECLARA | ε
VAR	->	TYPE ID (, ID)* '|' | TYPE ATR
TYPE	->	LITTERAE | NUMERUS
ATR	->	ID = (TEXTO | EXP) '|'
BLOCO	->	CMD BLOCO | ε
CMD	->	IN | OUT | LOOP | IF | EXP | ATR
IN	->	LEGERE [ID] '|'
OUT	->	SCRIBERE [ID | TEXTO] '|' | REDITUS [ID] '|'
LOOP	->	DUM [EXP OP EXP | BOOL] '(' BLOCO ')'
IF	->	SI [EXP OP EXP] '(' BLOCO ')' (ALIUD BLOCO)?
OP	->	LOGICO | ARITMETICO | RELACIONAL
LOGICO	->	AUT | ET | NO
ARITMETICO	->	N0 | N1

LOGICO	->	AUT | ET | NO
ARITMETICO	->	N0 | N1
N0	->	- | +
N1	->	* | /
RELACIONAL	->	> | < | >= | <=
EXP	->	TERM EXP'
EXP'	->	N0 EXP | ε
TERM	->	FATOR TERM'
TERM'	->	N1 TERM | ε
FATOR	->	NUMERO | ID | '(' EXP ')'
TEXTO	->	" (LETRA | NUMERO)* "
LETRA	->	(a..z | A..Z)*
ID	->	(a..z)(A..Z | a..z | 0..9)*
NUMERO	->	(0..9)+
BOOL	->	VERUM | FALSUS

Conjunto First e Follow

- O conjunto First são todos os terminais que podem ser o primeiro terminal para aquele dado símbolo. Essa informação é valiosa para o compilador, pois permite antecipar os possíveis símbolos iniciais de uma produção.
- O conjunto Follow de uma variável não-terminal representa os terminais que podem aparecer imediatamente após dado símbolo. Essa informação é crucial para determinar onde uma produção pode ser encerrada e como continuar a análise.

Conjunto First e Follow

Variável	First	Follow
MAIN	{SATUS}	{ $ }
DECLARA	{LITTERAE, NUMERUS, ε}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
VAR	{LITTERAE, NUMERUS}	{LITTERAE, NUMERUS, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
TYPE	{LITTERAE, NUMERUS}	{(a..z)(A..Z | a..z | 0..9)*}
ATR	{(a..z)(A..Z | a..z | 0..9)*}	{LITTERAE, NUMERUS, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
BLOCO	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*, ε}	{) }
CMD	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
IN	{LEGERE}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}

Conjunto First e Follow

OUT	{SCRIBERE, REDITUS}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
LOOP	{DUM}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
IF	{SI}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,)}
OP	{AUT, ET, NO, -, +, *, /, >, <, >=, <=}	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}
LOGICO	{AUT, ET, NO}	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}
ARITMETICO	{-, +, *, /}	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}
N0	{-, +}	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}
N1	{*, /}	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}
RELACIONAL	{>, <, >=, <=}	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}
EXP	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}	{|, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /}
EXP'	{-, +, ε}	{|, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /}

Conjunto First e Follow

TERM	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}	{[, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /}
TERM'	{*, /, ε}	{[, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /}
FATOR	{(0..9)+, (a..z)(A..Z | a..z | 0..9)*}	{LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /}
TEXTO	{"}	{LITTERAE, NUMERUS, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,]}
LETRA	{(a..z | A..Z)*}	{ " }
ID	{(a..z)(A..Z | a..z | 0..9)*}	{[, =, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /}
NUMERO	{(0..9)+}	{[, LEGERE, SCRIBERE, REDITUS, DUM, SI, (0..9)+, (a..z)(A..Z | a..z | 0..9)*,), AUT, ET, NO, +, -, *, /, >, <, >=, <=,], *, /, "}
BOOL	{VERUM, FALSUS}	{ [}

Fonte: Elaborada pelos autores, 2022

Compilador

- Um compilador é um programa que traduz o código-fonte de uma linguagem de programação para um código executável, passando por etapas como análise léxica, sintática, semântica e geração de código intermediário, culminando na produção de um código final em linguagem de máquina.

Requisitos Funcionais da Linguagem

Etapa	Função	Requisito Funcional
Análise Léxica	Verificar e reconhecer os Tokens	Deve receber o arquivo de entrada e identificar os tokens.
		Deve classificar e retornar os tokens para palavras-chave, identificadores, símbolos especiais, operadores lógicos, relacionais e aritméticos na linguagem.
		Deve possibilitar que os tokens produzidos sejam empregados nas fases seguintes do processo de compilação.
		Deve ignorar comentários, espaçamentos, quebras de linha e tabulações.
		Deve detectar erros léxicos, como palavras reservadas, símbolos e operadores.

Requisitos Funcionais da Linguagem

Análise Sintática	Verificar a estrutura da gramática	Deve analisar as configurações dos tokens em conformidade com a gramática estabelecida.
		Deve ser capaz de antecipar um único caminho a partir da leitura de um token.
		Deve detectar erros sintáticos, como a ordem dos tokens, símbolos mal estruturados e palavras reservadas fora de sequência.
Análise Semântica	Verificar os possíveis erros não tratados nas fases anteriores	Deve ser capaz de não declarar duas variáveis com o mesmo identificador.
		Deve verificar se a variável foi declarada antes de usar.
		Deve impedir que um carácter receba número e vise versa na declaração.

Requisitos Não Funcionais da Linguagem

Propriedade	Medida
Eficiência	O tempo de compilação deve ser proporcional à complexidade e tamanho do código.
Portabilidade	O compilador deve ser capaz de gerar código executável que seja compatível com diferentes plataformas e arquiteturas.
Usabilidade	O compilador deve fornecer mensagens de erros claros, contendo tipo de erro, sendo eles léxico, sintático, semântico e a linha que se encontra.
Confiabilidade	Ser robusto em relação a entradas inesperadas e garantir a consistência do processo de compilação.
Funcionalidade	Garantir que o compilador suporte recursão de maneira eficiente, permitindo a implementação de algoritmos e estruturas de dados recursivas.

Análise Léxica

- Ao construir o analisador léxico, a abordagem adotada envolveu a criação de um autômato finito determinístico diretamente no código.
- Foi implementado por meio de estruturas condicionais, que representam cada estado do autômato.
- Cada estado está associado a um padrão específico, como uma palavra-chave, identificador, operador ou símbolo, e as transições entre os estados são determinadas pela entrada do código-fonte.

Análise Sintática

- Ao construir o analisador sintático, foi necessário criar uma gramática da linguagem Rubrum, baseado na gramática preditiva.
- Este tipo de analisador é descendente (top-down), ou seja, inicia a análise a partir do primeiro símbolo da raiz da árvore e desce para as folhas, fazendo a leitura dos tokens da entrada da esquerda para a direita.

Análise Semântica

- A construção do analisador semântico foi feita para complementar a análise sintática e é incorporada como extensão no mesmo arquivo.
- Esta abordagem permite uma integração mais estreita entre as duas etapas do processo de compilação, simplificando a detecção de erros e realizando verificações semânticas mais avançadas.

Manual (mostrar no código)

- Estrutura do programa
- Comentários e símbolos obrigatórios
- Tipos de dados
- Declaração e Atribuição
- Entrada e Saída
- Comandos SI, ALIUD e DUM
- Erros

INSTITUTO FEDERAL
Minas Gerais
Campus Bambuí