UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Pablo Ferreira - 3480
Samuel Sena - 3494
Thomas Chang - 3052

TRABALHO PRÁTICO 2
COMPILADORES - CCF 441

Florestal
2021

Introdução

Nesta etapa do trabalho prático, ficamos encarregados de definir os conceitos e funcionamento de uma linguagem de programação. Para isso, realizaremos a definição dos tipos primitivos de dados aceitos pela nossa linguagem, além de comandos suportados, o paradigma de programação, palavras-chave e palavras reservadas. A gramática da linguagem, juntamente com tokens e padrões de lexemas são exemplos de elementos que também serão especificados.

Nome da Linguagem

O grupo escolheu o nome MAYBE para a linguagem, porque talvez funcione, talvez seja eficiente, talvez a gente consiga entregar tudo e talvez a gente passe na disciplina. A extensão dos arquivos escritos em MAYBE deverá ser “.may”.

Tipos de dados:

O grupo escolheu os seguintes tipos de dados:

Tipo de dados	**Palavra reservada**
Inteiro	int
Ponto flutuante	float
Caractere	char
String	string
Boolean	boolean

Comandos suportados:

Comando	**Ação**
=	Comando de atribuição
+	Comando aritmético de adição
-	Comando aritmético de subtração
*	Comando aritmético de multiplicação
/	Comando aritmético de divisão
mod	Comando aritmético de resto de divisão

==	Comando relacional de igualdade
>=	Comando relacional de superioridade ou igualdade
<=	Comando relacional de inferioridade ou igualdade
>	Comando relacional de superioridade
<	Comando relacional de inferioridade
<>	Comando relacional de diferença
not	Comando lógico de negação
or	Comando lógico de união
and	Comando lógico de intersecção

Paradigma de programação:

O paradigma de programação escolhido por nós para nossa linguagem de programação é o paradigma imperativo e estruturado. Dessa forma, teremos uma linguagem bem objetiva e com legibilidade alta. Nossa linguagem de programação também contará com o recurso de comentários (que serão desconsiderados durante o processo de compilação).

Palavras-chave e palavras reservadas:

As palavras-chave e palavras reservadas serão principalmente aquelas palavras utilizadas para a invocação de comandos durante a implementação do fluxo do código da linguagem. São elas:

- int
- float
- char
- string
- boolean
- void
- return
- mod
- not
- or
- and
- if
- elsif

- else
- while
- true
- false
- break
- continue

Lexemas e tokens

A tabela abaixo relaciona e explicita todos os tokens e lexemas a serem identificados e retornados pelo analisador léxico.

Token	**Lexema**	**Padrão**
INT	int	int
FLOAT	float	float
CHAR	char	char
STRING	string	string
VOID	void	void
RETURN	return	return
BREAK	break	break
CONTINUE	continue	continue
LE	<=	<=
GE	>=	>=
EQ	==	==
NE	<>	<>
NOT	not	not
OR	or	or
AND	and	and
IF	if	if

ELSIF	elsif	elsif
ELSE	else	else
WHILE	while	while
NUM_I	-?[0-9]+	(números inteiros)
NUM_F	-?[0-9]+.[0-9]+	(números reais)
ID	[a-zA-Z][a-zA-Z0-9]*	sequência de caracteres seguidos ou não por números
OP	mod	mod
OP	+	+
OP	-	-
OP	/	/
OP	*	*
ABREPARENTESES	(	(
FECHAPARENTESES	)	)
ABRECOLCHETES	[	[
FECHACOLCHETES	]	]
ABRECHAVES	{	{
FECHACHAVES	}	}
PVIRGULA	;	;
VIRGULA	,	,

Gramática da linguagem

Decidimos que nossa linguagem de programação irá desprezar espaços em branco e tabulações, dessa forma, as expressões deverão ser delimitadas com o uso do caractere terminal “;” (ponto e vírgula). Além disso, os escopos serão delimitados com os caracteres “{“ e “}”. Lembrando que o balanceamento deles será necessário para a correta sintaxe.

Para criar um comentario na linguagem, o mesmo deverá ser precedido dos caracteres “/*” e ao sucedido de “*/”. Dessa maneira, o compilador entenderá todo

conteúdo escrito entre tais delimitadores deverá ser desconsiderado para o processo de compilação.

A forma de declaração de uma variável seguirá a seguinte máscara:

Tipo *Nome_da_variável;*

Sendo que, inicialmente o tipo da variável deverá ser definido, e em seguida, o nome do identificador.

A forma de declarar um subprograma ou função também segue tal formula:

```
Tipo Nome_da_funcao(Tipo1 param1,Tipo2 param2...){
        /* Corpo da função com retorno */
}
```

Sendo que, inicialmente o tipo do retorno é definido, em seguida é escrito o nome ou identificador da função, a frente os parâmetros deverão ser declarados entre parênteses e separados por vírgulas e o escopo é definido pelas chaves ({,}).

O conteúdo de strings deve ser escrito entre aspas duplas e os caracteres devem ser escritos entre aspas simples (para atribuir conteúdos em variáveis do tipo string e carácter).

Com isso, temos a seguinte gramática para a linguagem MAYBE:

expr
: expr '**+**' expr ;
| expr '**-**' expr ;
| expr '*****' expr ;
| expr '**/**' expr ;
| expr '**mod**' expr ;
| expr '**==**' expr ;
| expr '**<>**' expr ;
| expr '**<=**' expr ;
| expr '**>=**' expr ;
| expr '**>**' expr ;
| expr '**<**' expr ;
| '**(**' expr '**)**' ;
| '**[**' expr '**]**' ;
| '**{**' expr '**}**' ;
| expr '**or**' expr ;
| expr '**and**' expr ;

```
        | expr ‘not’ ;
        | expr ‘=’ expr ;
        | term ;

term
        : -?[digit]+ ;
        | +?[digit]+ ;
        | +?[digit]+.[digit]+ ;
        | -?[digit]+.[digit]+ ;
        | [letter]+ ;
        | return ;
        | continue ;
        | break ;
        | int ;
        | float ;
        | string ;
        | char ;
        | boolean ;
        | void ;
        | ‘;’ ;
        | ‘,’ ;

while
        : while '(' expr ')' '{' stmt '}' ;

conditional
        : if ‘(‘ expr ‘)’ ‘{‘ stmt ‘}’ ;
        | elsif ‘(‘ expr ‘)’ ‘{‘ stmt ‘}’;
        | else ‘{‘ stmt ‘}’ ;
stmt
        :  expr expr expr ;
        | term stmt ;
        | expr ;
digit
        : "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
letter
        : [a-zA-Z];
```

Gerador de analisador léxico

O nosso analisador léxico foi gerado utilizando o gerador de analisador léxico Flex. Para compilá-lo basta executar “flex lex.l” e em seguida “gcc lex.yy.c”. O binário final “a.out” consiste em nosso analisador léxico. Para executar a análise léxica de um arquivo de código fonte, basta executar o comando na seguinte forma: “./a.out < *nome_arquivo_de_entrada”.*

Realizamos a implementação de cinco arquivos de entrada com o intuito de testar se todos os padrões aqui definidos estão sendo de fato reconhecidos e tendo seus tokens devidamente retornados na tela. Os arquivos de entrada estão nomeados de 1 a 5 e estão na mesma pasta que o arquivo lex.l. As figuras abaixo exibem saídas parciais obtidas para a execução dos arquivos:

Figura 1 - Análise léxica do arquivo entrada4.may.

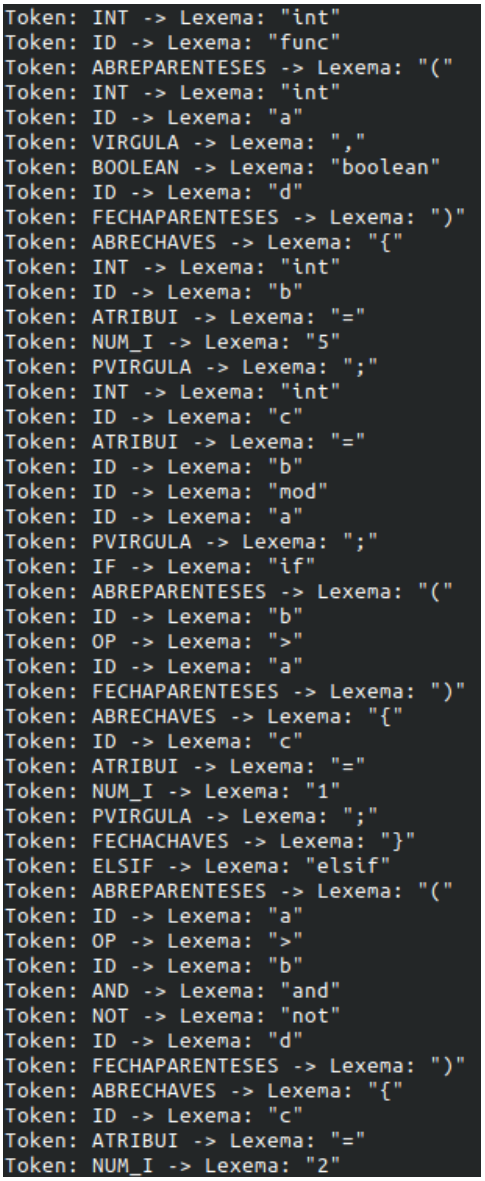


Figura 2 - Análise léxica do arquivo entrada5.may.

```
Token: VOID -> Lexema: "void"
Token: ID -> Lexema: "laco"
Token: ABREPARENTESES -> Lexema: "("
Token: FECHAPARENTESES -> Lexema: ")"
Token: ABRECHAVES -> Lexema: "{"
Token: INT -> Lexema: "int"
Token: ID -> Lexema: "i"
Token: ATRIBUI -> Lexema: "="
Token: NUM_I -> Lexema: "0"
Token: PVIRGULA -> Lexema: ";"
Token: WHILE -> Lexema: "while"
Token: ABREPARENTESES -> Lexema: "("
Token: ID -> Lexema: "i"
Token: OP -> Lexema: "<"
Token: NUM_I -> Lexema: "25"
Token: FECHAPARENTESES -> Lexema: ")"
Token: ABRECHAVES -> Lexema: "{"
Token: IF -> Lexema: "if"
Token: ABREPARENTESES -> Lexema: "("
Token: ID -> Lexema: "i"
Token: ID -> Lexema: "mod"
Token: NUM_I -> Lexema: "2"
Token: EQ -> Lexema: "=="
Token: NUM_I -> Lexema: "0"
Token: FECHAPARENTESES -> Lexema: ")"
Token: ABRECHAVES -> Lexema: "{"
Token: ID -> Lexema: "i"
Token: ATRIBUI -> Lexema: "="
Token: ID -> Lexema: "i"
Token: OP -> Lexema: "+"
Token: NUM_I -> Lexema: "1"
Token: PVIRGULA -> Lexema: ";"
Token: BREAK -> Lexema: "break"
Token: PVIRGULA -> Lexema: ";"
Token: FECHACHAVES -> Lexema: "}"
Token: ELSE -> Lexema: "else"
Token: ABRECHAVES -> Lexema: "{"
Token: ID -> Lexema: "i"
Token: ATRIBUI -> Lexema: "="
Token: ID -> Lexema: "i"
Token: OP -> Lexema: "+"
Token: NUM_I -> Lexema: "5"
Token: PVIRGULA -> Lexema: ";"
Token: CONTINUE -> Lexema: "continue"
Token: PVIRGULA -> Lexema: ";"
Token: FECHACHAVES -> Lexema: "}"
Token: FECHACHAVES -> Lexema: "}"
Token: FECHACHAVES -> Lexema: "}"
```

Figura 3 - Análise léxica do arquivo entrada2.may.

```
Token: INT -> Lexema: "int"
Token: ID -> Lexema: "main"
Token: ABREPARENTESES -> Lexema: "("
Token: FLOAT -> Lexema: "float"
Token: ID -> Lexema: "j"
Token: FECHAPARENTESES -> Lexema: ")"
Token: ABRECHAVES -> Lexema: "{"
Token: STRING -> Lexema: "string"
Token: ID -> Lexema: "b"
Token: ATRIBUI -> Lexema: "="
Token: STR -> Lexema: ""Ola mundo""
Token: PVIRGULA -> Lexema: ";"
Token: FLOAT -> Lexema: "float"
Token: ID -> Lexema: "c"
Token: ATRIBUI -> Lexema: "="
Token: NUM_F -> Lexema: "3.14"
Token: PVIRGULA -> Lexema: ";"
Token: ID -> Lexema: "retrun"
Token: ABREPARENTESES -> Lexema: "("
Token: ID -> Lexema: "c"
Token: OP -> Lexema: "+"
Token: ID -> Lexema: "j"
Token: FECHAPARENTESES -> Lexema: ")"
Token: VIRGULA -> Lexema: ","
Token: ID -> Lexema: "b"
Token: PVIRGULA -> Lexema: ";"
Token: FECHACHAVES -> Lexema: "}"
```

Figura 4 - Análise léxica do arquivo entrada3.may.

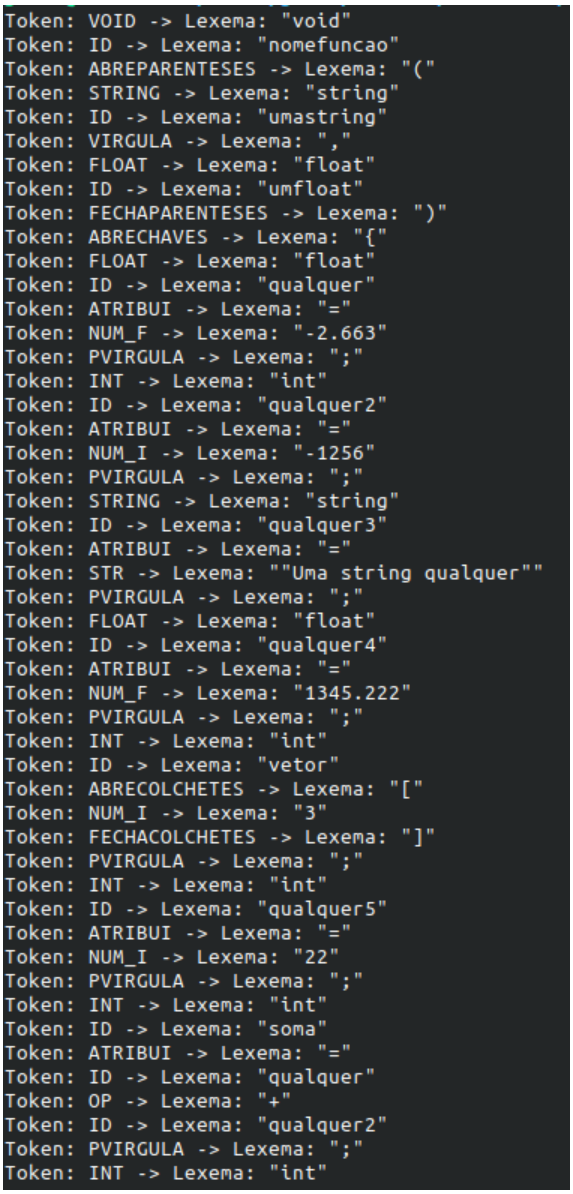
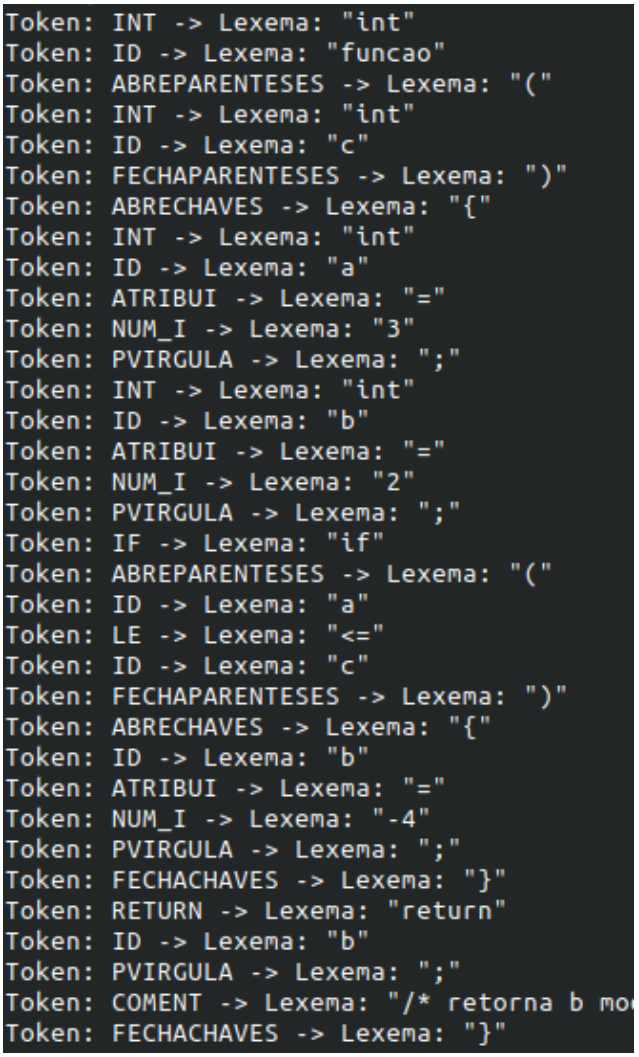


Figura 5 - Análise léxica do arquivo entrada.may.



Conclusão

A realização deste trabalho foi de grande ajuda para uma melhor compreensão do processo inicial de criação e definição de um compilador de uma linguagem de programação. Esperamos aprender ainda mais sobre o assunto para que possamos, nas etapas seguintes, realizar as modificações necessárias para um funcionamento mais correto e assim implementar de maneira melhor nossa linguagem de programação.

Todas as dúvidas que tivemos durante o processo de desenvolvimento desta parte do trabalho foram esclarecidas com diversas pesquisas pela Web, e as principais fontes de tais esclarecimentos foram devidamente referenciadas.

Referências

DEGENER, Jutta. ANSI C Yacc grammar. 1995. Disponível em: https://www.lysator.liu.se/c/ANSI-C-grammar-y.html. Acesso em 21 set. 2021.

GRAMÁTICAS: fundamentos e exemplos. Disponível em: https://www.ime.usp.br/~fmario/mac122-15/gramatica.html. Acesso em: 15 set. 2021.

QUADROS, Everton Quadros Everton. **Software Developer © 2020 Implementando um analisador léxico usando o Flex**. Disponível em: https://eqdrs.github.io/compilers/2019/09/08/implementando-um-analisador-lexico-usando-o-flex.html. Acesso em: 15 set. 2021.

FULL Grammar specification. Disponível em: https://docs.python.org/3/reference/grammar.html. Acesso em: 16 set. 2021.