UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

EWELLY FABIANE CUNHA DE SOUSA

PROJETO FINAL DE CONSTRUÇÃO DE COMPILADORES

Boa Vista, RR
2022

EWELLY FABIANE CUNHA DE SOUSA

PROJETO FINAL DE CONSTRUÇÃO DE COMPILADORES

Trabalho apresentado como requisito parcial para obtenção de nota na disciplina de Construção de compiladores ofertada pelo curso de Ciência da Computação da Universidade Federal de Roraima.

Prof.: Dr. Luciano Ferreira Silva

Boa Vista, RR

2022

1 RELATÓRIO

Para desenvolver foi utilizado a linguagem python por ter ferramentas convenientes para a construção do programa. Foram usadas, basicamente, estruturas de dados de lista e dicionário e desenvolvidas as etapas: separação de lexemas, análise léxica, análise sintática e análise semântica. Todas as operações que gerenciam os analisadores, sejam léxica, sintática, semântica e operações com as informações que cada um gera é realizada no arquivo “main.py”.

1. Funcionamentos e comandos da linguagem fonte:

A linguagem escolhida foi o C (modificado). Onde nesta linguagem temos:

I. Aceita as operações aritméticas: soma ‘+’ e subtração ‘-’.

II. Atribuição com o operador ‘ = ‘ e para determinar precedências são utilizados parênteses ‘ (‘ e ‘) ’.

III. As linhas são delimitadas por ponto e vírgula ‘ ; ‘.

IV. Caracteres são separados por espaço e vírgulas.

VI. Sua estrutura pode ter ou não ter funções, ou seja, apenas escrever as instruções.

VII. Possui as palavras reservadas listadas abaxo:

"main", "scanf", "printf", "return", "if", "else", "elif","for", "do", "while", "int", "char", "or"

Operador de soma:	+
Operador de subtração:	-
Operador de atribuições	=
Operador para determinar precedências:	‘ (‘ e ‘) ’
Delimitador de linhas	;
Separador de caracteres. Ex: int a,b, c	‘ ‘ e ‘,’
Funções	if(a==b){}
Instruções	r#esult(5);

2. Analisador léxico.

Separação de lexemas

O programa lê o arquivo de entrada, coloca cada sentença adicionando em um vetor(lexema) usando append(), separado os quando há um espaço, através do split(), também considera sinais que podem estar juntos das variáveis e os separa com espaço. Depois o vetor lexema é reconstruído transformando-o em um matriz de forma a facilitar posteriormente a inserção de colunas para cada lexema do vetor.

Depois cada linha da matriz é lida, levando em consideração que a primeira coluna é a qual será rotulada. As rotulações são separadas em: $, PR, ID e EXP. Onde $ trata-se do delimitador de linhas: “;”, PR o rótulo para palavras reservadas, ID o rótulo para variáveis em geral. E EXP contém todas as expressões possíveis a serem rotuladas, considerando que suas correções serão feitas em outra etapa. Para fins de conhecimento, no final da execução desta etapa é gerado um arquivo com as separações:

```
int             PR
f#ibo           ID
(               (
int             PR
n#um            ID
)               )
{               {
if              PR
(               (
n#um            ID
=               =
1               NUM
or              PR
n#um            ID
=               =
0               NUM
)               )
{               {
return          PR
n#um            ID
;               $
}               }
}               }
int             PR
a#we1           ID
=               =
f#ibo           ID
(               (
5               NUM
)               )
;               $
a#we1           ID
=               =
2               NUM
;               $
(               (
r#esult         ID
=               =
2               NUM
```

Nos casos de uso de sinais como parênteses, colchetes, e indicação de texto, observa-se que é rotulado separadamente dos dados de onde são usados. Para os casos de teste, foram usadas as seguintes operações:

codigo.txt

```
1   int f#ibo(int n#um){
2       if(n#um = 1 or n#um=0){
3           return n#um;
4       }
5   }
6
7   int a#we1 = f#ibo(5);
8   a#we1 = 2;
9   (r#esult = 2)=4;
10
```

Análise léxica

A gramática regular usada identifica se um nome de variável é válida, assim usa somente os rótulos com ID da etapa anterior, o primeiro estado(E1) analisa se o primeiro caractere é uma letra o segundo estado (E2) verifica se o próximo caractere é o sinal especial “#”, após isso o próximo estado já é o estado final(E3) onde há um self-loop que aceita letras, números e o caracter especial #.

alfabeto{a, b,#}, tal que a = [a-z] # [a-Z,0-9,#]
L = a#(a | b | #) *
Segue a matriz de estados:

	E0	E1	E2
A (alfanuméricos)	1	777	2
B (#)	777	2	2
C (números)	777	777	2

Análise sintática

É usado o analisador preditivo com a gramática (S). A gramática (S) faz um id receber um valor, que pode ser até mesmo outro id (i = i), ou uma expressão complementando a gramática S (i = v). A gramática também aceita atribuições em sequência, neste caso, a expressão, se houver, deve estar somente na última atribuição (ex: i=i=v).

Gramática S
Vt = {=, (,), v}, Vn = {E, F, M}, símbolo sentencial E
P1: E -> MF
P2: F -> =MF
P3: F -> &
P4: M -> (E)
P5: M -> v

Tabela Sintática:

	+	(	)	v	$
E	9	1	9	1	9
F	2	9	3	9	5
M	9	4	9	5	9

Para a árvore sintática do analisador preditivo, da mesma forma, também utiliza uma lista A com cada produção gerada pela análise da sentença. Mas desta vez a árvore precisa ser criada de forma precedente, sendo assim utiliza se a estratégia de recursão, onde partindo do símbolo sentencial, cria se um nodo para cada símbolo, e o próximo símbolo da lista são tidos como filho de deste nodo, e é invocado a recursão destes, até que encontre um símbolo terminal, desta forma cada filho é retornado para o nodo anterior, assim formando a árvore. É usando uma pilha auxiliar para percorrer a lista, e desempilhar caso o símbolo no topo seja um terminal, assim os nodos folhas são conectados corretamente na árvore.

Análise semântica

Nesta etapa , foi feita a verificação de tipos, levando em consideração que o código da entrada já esteja adaptado para ser percorrido por esta verificação. Para verificar, utiliza se estrutura de dados lista para identificar sinais básicos, e dicionário para verificação do tipo na variável.

```
# tipos de variaveis
tipo = {'int': int, 'char': str, 'void': None}
sinais = ['+', '-', '%', '=']
```

(Também é usando um dicionário que armazenará as variáveis declaradas, além da lista que conterá cada elemento de cada linha do código)

A seguir a lista que contém cada trecho do código da entrada é percorrida, para identificar as declarações, para então serem adicionadas ao dicionário de declarações; estas são identificadas a partir do reconhecimento de um elemento em tipo, obtendo assim a variável e seu tipo, é armazenando também a linha do código onde ela foi declarada; também é adicionado ao dicionário de declarações, funções e suas variáveis invocadas; a forma de como elas são identificadas é tratado posteriormente, para as funções é adicionado a posição onde termina uma função, quando o símbolo ‘}’ é identificado; estando esta posição, igual a posição de início quando foi identificado inicialmente, isto para toda declaração; é usado uma pilha auxiliar para identificar a função.

Após uma declaração de variável ser reconhecida, ela é removida da lista dos trechos; esta lista posteriormente é usada para analisar expressões e usa o dicionário de declarações para reconhecer as variáveis. Nesta parte também é tratado os casos de declaração com atribuição na mesma linha, neste caso usa se uma pilha auxiliar que adiciona na lista a atribuição após a declaração ser removida, na mesma posição da lista.