Instituto de Ciências Matemáticas e de Computação
Departamento de Sistemas de Computação

SCC0250 - Computação Gráfica

Trabalho 1 - Analisador Léxico

Lucas Sobral Fontes Cardoso 8957176
Werik Amaral Fachim 7656512

São Carlos, 27 de Abril de 2019.

Decisões

1. **Tokens da forma <palavra_reservada, palavra_reservada>**

Foi optada que os tokens para palavras reservadas assumissem a forma <palavra_reservada, palavra_reservada>, pois consideramos que este formato traz a mesma informação que utilizando simb_palavra_reservada, entretanto o faz de forma mais resumida.

2. **Tabelas hash para armazenamento das palavras reservadas**

Utilizou-se uma tabela hash para armazenar as palavras reservadas com função linear para o cálculo de índice na tabela, pois isto permite que a busca seja feita em tempo constante(O(1)), cuja propriedade é bastante importante para a velocidade da execução do analisador léxico.

3. **Previsão de erros**

Mostrar a particularidade do erro para o usuário de um compilador é bastante importante, pois permite facilitar a correção dos mesmos. Com isso em mente, procurou-se mostrar os erros de forma bastante específica ao usuário, criando casos de correspondência para erros específicos, e mantendo a seguinte linha para os erros que não foram previstos:

```
. {fprintf(stdout, "%s - erro", yytext);}
```

Descrição da especificação do analisador léxico

O analisador léxico foi feito utilizando a ferramenta Lex, no arquivo analisadorLexico.l. Este arquivo será então compilado para gerar o lex.yy.c, que irá descrever as funções yylex, yytext e yyleng, que são incluídas como externas na biblioteca lexer.h para uso posterior na função principal.

No arquivo .l, é descrito o código de números inteiros que são utilizadas como retorno da função yylex e as expressões regulares que correspondem a cada token. Então, na função principal, cada vez que é chamada a função yylex retorna o valor de um token que foi lido, além de yytext receber a cadeia que foi lida.

Na biblioteca lexer.h definimos uma função que converte o inteiro que yylex retorna na string que corresponde o tipo de token lido, que é então mostrado na saída padrão pela função principal. A única exceção ocorre para tokens de identificadores, que antes de irem para a saída verifica-se se eles correspondem a uma palavra reservada ou excedem o limite de tamanho, então serão retornados tokens de palavras reservadas ou erro, respectivamente.

Compilação

A compilação é gerenciada pelo arquivo makefile, que possui os seguintes comandos:

```
make - Compila os arquivos de código e gera um executável a.out.
make test - Executa utilizando o arquivo input.txt como entrada.
make run - Executa utilizando a entrada padrão como entrada.
```

O comando make utiliza os comandos da diretiva all: `flex analisadorLexico.l` para gerar o arquivo lex.yy.c que será então utilizado no comando `gcc -lm -o a.out *.c` que gera o executável a partir do .c gerado pelo lex e outras bibliotecas utilizadas.

Exemplos de execução

```
{este programa testa o compilador}
{testes com operadores}
+
-
*
/
=
<
>
<
<
>
{testes com simbolos}
.
:
(
)
;
,
: ,
{testes com numeros}
10
1
94.36
13.
1.1
10000.1
{testes com identificadores}
_window
numeroIdentificacao
_lucas
1_lucas
1a
readd
{testes com caracteres invalidos}
numero@com
_lucas@
@number
{testes com numeros reais mal formados}
3.a2
3.3a2

program lalg;
{entrada}
var a: integer;
begin
read(a, @, 1);
end.
```

```
wrk@Mercur:~/Dropbox/USP/Disciplinas/Compiladores/t1/AnalisadorLALG$ make test
./a.out < input.txt
+ - op_soma
- - op_subt
* - op_mult
/ - op_div
= - op_igual
< - op_menor
> - op_maior
<= - op_menor
<> - op_menor
>= - op_maior
. - simb_ponto
: - simb_dois_pontos
( - simb_abrir_parenteses
) - simb_fechar_parenteses
; - simb_ponto_virgula
, - simb_virgula
:= - simb_dois_pontos
, - simb_virgula
10 - numero_inteiro
1 - numero_inteiro
94.36 - numero_real
13 - numero_inteiro
. - simb_ponto
1.1 - numero_real
10000.1 - numero_real
_window - identificador
numeroIdentificacao - identificador
_lucas - identificador
1_lucas - error - identificador mal formado
1a - error - identificador mal formado
readd - identificador
numero - identificador
@ - erro - caractere invalido
com - identificador
_lucas - identificador
@ - erro - caractere invalido
@ - erro - caractere invalido
number - identificador
3.a2 - erro - numero real mal formado
3.3a2 - erro - numero real mal formado
program - program
lalg - identificador
; - simb_ponto_virgula
var - identificador
a - identificador
: - simb_dois_pontos
integer - integer
; - simb_ponto_virgula
begin - begin
read - read
( - simb_abrir_parenteses
a - identificador
, - simb_virgula
@ - erro - caractere invalido
, - simb_virgula
1 - numero_inteiro
) - simb_fechar_parenteses
; - simb_ponto_virgula
```

```
{este programa testa o compilador}

program lalg;
{entrada}
var a, minha_var: integer;
var pedra,tesoura: const;
var raeel: real;
beggin
readd(a, @, 1);
whille(minha_var,1.23);
if(var > 10
procedure
elseeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeerrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
writee(minha_var,"string teste")
end.

```

```
wrk@Mercur:~/Dropbox/USP/Disciplinas/Compiladores/t1/AnalisadorLALG$ make test
./a.out < input3.txt
program - program
lalg - identificador
; - simb_ponto_virgula
var - identificador
a - identificador
, - simb_virgula
minha_var - identificador
: - simb_dois_pontos
integer - integer
; - simb_ponto_virgula
var - identificador
pedra - identificador
, - simb_virgula
tesoura - identificador
: - simb_dois_pontos
const - const
; - simb_ponto_virgula
var - identificador
raeel - identificador
: - simb_dois_pontos
real - real
; - simb_ponto_virgula
beggin - identificador
readd - identificador
( - simb_abrir_parenteses
a - identificador
, - simb_virgula
@ - erro - caractere invalido
, - simb_virgula
1 - numero_inteiro
) - simb_fechar_parenteses
; - simb_ponto_virgula
whille - identificador
( - simb_abrir_parenteses
minha_var - identificador
, - simb_virgula
1.23 - numero_real
) - simb_fechar_parenteses
; - simb_ponto_virgula
if - if
( - simb_abrir_parenteses
var - identificador
> - op_maior
10 - numero_inteiro
procedure - procedure
elseeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeerrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
rrrrrrrrrrrrrrrrrrrrrrrrrr - erro - identificador muito grandeelseeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeerrr
rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr - procedure
writee - identificador
( - simb_abrir_parenteses
minha_var - identificador
, - simb_virgula
" - errostring - do
teste - identificador
" - erro) - simb_fechar_parenteses
end - end
. - simb_ponto
```

```
1  {este programa testa o compilador}
2
3  program lalg;
4  {entrada}
5  var a: integer;
6  var avariavel: const;
7  var raeeal: real;
8  begin
9  read(a, @, 1);
10 while(0)
11 if procedure
12 end.
```

```
wrk@Mercur:~/Dropbox/USP/Disciplinas/Compiladores/t1/AnalisadorLALG$ make test1
./a.out < input1.txt
program - program
lalg - identificador
; - simb_ponto_virgula
var - identificador
a - identificador
: - simb_dois_pontos
integer - integer
; - simb_ponto_virgula
var - identificador
avariavel - identificador
: - simb_dois_pontos
const - const
; - simb_ponto_virgula
var - identificador
raeeal - identificador
: - simb_dois_pontos
real - real
; - simb_ponto_virgula
begin - begin
read - read
( - simb_abrir_parenteses
a - identificador
, - simb_virgula
@ - erro - caractere invalido
, - simb_virgula
1 - numero_inteiro
) - simb_fechar_parenteses
; - simb_ponto_virgula
while - while
( - simb_abrir_parenteses
0 - numero_inteiro
) - simb_fechar_parenteses
if - if
procedure - procedure
end - end
. - simb_ponto
```

```
1  {este programa testa o compilador}
2
3  program lalg;
4  {entrada}
5  var a, minha_var: integer;
6  var pedra,tesoura: const;
7  var raeel: real;
8  begin
9  read(a, @, 1);
10 while(minha_var,1.23);
11 if(var > 10)
12 procedure
13 else
14 write(minha_var,"string teste")
15 end.
16
```

```
wrk@Mercur:~/Dropbox/USP/Disciplinas/Compiladores/t1/AnalisadorLALG$ make test2
./a.out < input2.txt
program - program
lalg - identificador
; - simb_ponto_virgula
var - identificador
a - identificador
, - simb_virgula
minha_var - identificador
: - simb_dois_pontos
integer - integer
; - simb_ponto_virgula
var - identificador
pedra - identificador
, - simb_virgula
tesoura - identificador
: - simb_dois_pontos
const - const
; - simb_ponto_virgula
var - identificador
raeel - identificador
: - simb_dois_pontos
real - real
; - simb_ponto_virgula
begin - begin
read - read
( - simb_abrir_parenteses
a - identificador
, - simb_virgula
@ - erro - caractere invalido
, - simb_virgula
1 - numero_inteiro
) - simb_fechar_parenteses
; - simb_ponto_virgula
while - while
( - simb_abrir_parenteses
minha_var - identificador
, - simb_virgula
1.23 - numero_real
) - simb_fechar_parenteses
; - simb_ponto_virgula
if - if
( - simb_abrir_parenteses
var - identificador
> - op_maior
10 - numero_inteiro
) - simb_fechar_parenteses
procedure - procedure
else - else
write - write
( - simb_abrir_parenteses
minha_var - identificador
, - simb_virgula
" - errostring - do
teste - identificador
" - erro) - simb_fechar_parenteses
end - end
. - simb_ponto
```