11/12/2024

Trabalho de Compiladores

UFES

Ciência da Computação
Prof. Rodrigo Freitas Silva

Um Analisador Léxico para a linguagem C--

1) Implemente um analisador léxico para a linguagem C-- cuja gramática é descrita no Anexo I. Siga as instruções e observações abaixo:

a) Você deverá criar um IDE para o desenvolvimento e compilação dessa linguagem

b) O programa deverá diferenciar letras maiúsculas de letras minúsculas

i. Identificadores deverão conter até 31 caracteres (C89)

ii. Os identificadores podem usar letras maiúsculas ou minúsculas (de A a Z, sem acentos), números e o caractere sublinhado (_), mas o primeiro caractere deve ser uma letra ou o sublinhado

iii. Não é permitido caracteres de conexão nos identificadores ($%@#!? ...), exceto o sublinhado (_)

c) As palavras chave são reservadas pela linguagem

d) Os comentários deverão ser removidos do programa. Os comentários nesta linguagem são delimitados por // ou por **/*** ***/** e escritos como no exemplo:

```
// Este é um comentário de Linha
/*  Este é um
    Comentário de
Várias Linhas   */
```

e) Os espaços em branco também são automaticamente removidos pelo Analisador Léxico

f) Quando for encontrado um erro léxico, seu analisador deverá indicar a linha e a coluna do programa no qual o erro foi encontrado, assim como sublinhar de vermelho o erro no programa fonte

i. Você deve TENTAR encontrar esses erros léxicos não somente após o usuário mandar compilar o programa, mas também enquanto ele estiver escrevendo

g) Lembre-se de fazer seu compilador reconhecer automaticamente todos os tokens conhecidos antecipadamente (palavras reservadas, sinais de pontuação, operadores). Inclua os tokens encontrados no programa fonte na Tabela de Símbolos.

h) Você deverá escolher pelo menos uma estratégia de recuperação de erros

i) Ao final (na data marcada com o professor), **você deverá apresentar seu trabalho e entregar um relatório** descrevendo o funcionamento do seu IDE e do seu programa, como foi o desenvolvimento do seu trabalho, o diagrama relacionando os módulos do seu programa, uma tabela de (tokens, lexemas, expressões regulares, atributos e valor) para cada token encontrado na linguagem.

j) Caso haja algum erro na gramática que inviabilize a implementação de um programa escrito em C, ajuste a gramática o quanto for necessário para concertá-la.

2) Observações:

- As estruturas de controle **for** e **while** deverão ser incorporadas a gramática por você na análise sintática
- Pode haver instruções com somente um **ID ;**
  - Avisar com '*Warning*' que a instrução não tem efeito nenhum na instrução na Analise Sintática
- Poderão haver declarações de variáveis e a criação de subprogramas em qualquer lugar do programa fonte, inclusive um dentro do outro
- Ponto extra: incluir novas funcionalidades como tipo estrutura (**struct**), tipo enumeração (**enum**) ou #include "biblioteca"
- Não permitir que o índice de um vetor seja um literal

Anexo I

<programa> → *<especificador>* *<tipo>* **ID** *<programa2>* | **#define ID NUM** *<CRLF>* *<programa>* | **ϵ**

<especificador> → **AUTO** | **STATIC** | **EXTERN** | **CONST** | **ϵ**

<tipo> → **VOID** | **CHAR** | **FLOAT** | **DOUBLE** | **SIGNED** *<inteiro>* | **UNSIGNED** *<inteiro>* | *<inteiro>*

<inteiro> → **SHORT** | **INT** | **LONG**

<programa2> → ; *<programa>* | **[NUM]** ; *<programa>* |
(*<listaParametros>***)** *<bloco>* *<programa>* |
, *<listaID>* *<programa>*

<listaID> → ID *<declaracaoParam2>* *<listaIDTail>*

<listaIDTail> → ; | , *<listaID>*

<listaParametros> → *<listaParamRestante>* | **ϵ**

<listaParamRestante> → *<declaracaoParam>* *<declParamRestante>*

<declaracaoParam> → *<tipo>* **ID** *<declaracaoParam2>*

<declaracaoParam2> → **[NUM]** | **ϵ**

<declParamRestante> → , *<listaParamRestante>* | **ϵ**

<bloco> → **{** *<conjuntoInst>* **}** | ; *<conjuntoInst>*

<*conjuntoInst*> → <*programa*> <*conjuntoInst*> | <*instrucoes*> <*conjuntoInst*> | **ϵ**

<*instrucoes*> → **ID** <*expressao*> **;** | **RETURN** <*expr*> **;** | **PRINTF (** <*expr*> **) ;** |
SCANF (**ID**) **;** | **BREAK ;** | **IF** (<*expr*>) <*instrucoes*> <*instrucoesIf*>

<*instrucoesIf*> → **ELSE** <*instrucoes*> | **ϵ**

<*expressao*> → <*atribuicao*> | **[** <*expr*> **]** <*atribuicao*> | **(** *exprList* **)** | **ϵ**

<*atribuicao*> → <*operadorAtrib*> <*expr*>

<*operadorAtrib*> → **=** | ***=** | **/=** | **%=** | **+=** | **-=**

<*expr*> → <*exprAnd*> <*exprOr*>

<*exprList*> → <*expr*> <*exprListTail*> | **ϵ**

<*exprListTail*> → **,** <*exprList*> | **ϵ**

<*exprOr*> → **OR** <*exprAnd*> <*exprOr*> | **ϵ**

<*exprAnd*> → <*exprEqual*> <*exprAnd2*>

<*exprAnd2*> → **AND** <*exprEqual*> <*exprAnd2*> | **ϵ**

<*exprEqual*> → <*exprRelational*> <*exprEqual2*>

<*exprEqual2*> → **==** <*exprRelational*> <*exprEqual2*> | != <*exprRelational*> <*exprEqual2*> | **ϵ**

<*exprRelational*> → <*exprPlus*> <*exprRelational2*>

<*exprRelational2*> → **<** <*exprPlus*> <*exprRelational2*> | **<=** <*exprPlus*> <*exprRelational2*> |
> <*exprPlus*> <*exprRelational2*> | **>=** <*exprPlus*> <*exprRelational2*> | **ϵ**

<*exprPlus*> → <*exprMult*> <*exprPlus2*>

<*exprPlus2*> → **+** <*exprMult*> <*exprPlus2*> | **-** <*exprMult*> <*exprPlus2*> | **ϵ**

<*exprMult*> → <*exprUnary*> <*exprMult2*>

<*exprMult2*> → ***** <*exprUnary*> <*exprMult2*> | **/** <*exprUnary*> <*exprMult2*> | **ϵ**

<*exprUnary*> → **+** <*exprParenthesis*> | **-** <*exprParenthesis*> | <*exprParenthesis*>

<*exprParenthesis*> → (<*expr*>) | <*primary*>

<*primary*> → **ID** <*primaryID*> | **NUM** | **LITERAL**

<*primaryID*> → **[** <*primary*> **]** | **(** <*exprList*> **)** | **ϵ**

NUM → <*dígitos*> <*optFracionaria*>

<*dígitos*> → <*dig*> <*dígitos*>*

<*dig*> → [0-9]

<*optFracionaria*> → .<*dígitos*> | **ϵ**

ID → <*letra*><*letras*>

<*letra*> → [A-Za-z_]

<*letras*> → [<*letra*> | <*dig*>]*

LITERAL → ‘[<*letra*> | <*dig*>]*’

<*CRLF*> → {indica termino e quebra de linha}