

LINGUAGENS FORMAIS E COMPILADORES
AP2 – TRABALHO EM GRUPO
PROF. CLAYTON JONES ALVES DA SILVA

Condições gerais:

1. O trabalho corresponde a segunda avaliação bimestral.
2. O trabalho deve ser realizado e submetido no grupo.
3. Data de entrega e apresentação do trabalho: **17 de novembro de 2022.**

Dados do problema:

Definimos aqui uma linguagem de programação denominada C--, que é uma linguagem apropriada para um projeto de compilador por ser mais complexa que a linguagem TINY.

Trata-se essencialmente de um subconjunto de C, mas sem algumas partes importantes, o que justifica seu nome.

Neste documento listamos as convenções léxicas da linguagem, incluindo uma descrição dos *tokens*, assim como a sua gramática.

CONVENÇÕES LÉXICAS DE C - -

Palavras-chave

As palavras-chave da linguagem são as seguintes:

{else if int float return void while main read print}

Todas as palavras-chave são reservadas e devem ser escritas com caixa baixa

Símbolos especiais

Os símbolos especiais são os seguintes:

{'+', '-', '', '/', '<', '<=', '>', '>=', '=', '!=', '/', '*', '/', ',', '{', '}', '.', '(', ')', ';'}*

Tokens

Utiliza os *tokens* SIMBOLO e RESERVADA, para descrever os símbolos especiais e palavras-chave. Utiliza, ainda, os *tokens* ID, NUMINT e NUMFLOAT, que são definidos pelas expressões regulares a seguir

ID = letra (letra|dígito)*

NUMINT = dígito dígito*

NUMFLOAT = NUMINT . NUMINT*

letra = al...|z|Al.. |Z

dígito = 0|1|...|9

Existe diferença entre caixa baixa e caixa alta.

Espaços em branco

Compostos por brancos, mudanças de linha e tabulações. O espaço em branco é ignorado, exceto como separador de IDs, NUMs e palavras-chave.

Comentários

São cercados pela notação usual de C `/*...*/`. Os comentários podem ser colocados em qualquer lugar que possa ser ocupado por um espaço em branco e só podem incluir uma linha. Comentários não podem ser aninhados.

GRAMÁTICA DA C--

1. *programa* → *lista-fun* **main () { decl-sequência }**
2. *lista-fun* → *lista-fun fun | fun*
3. *fun* → *fun-decl { decl-sequência | decl-sequência return identificador }*
4. *fun-decl* → *tipo-idt identificador (parametros)*
5. *parâmetros* → *param-lista | void*
6. *param-lista* → *param-lista , param*
7. *param* → *tipo-idt identificador*
8. *decl-sequência* → *declaração | declaração ; declaração*
9. *declaração* → *var-decl | if-decl | atribuição-decl | while-decl*
 - i. *read-decl | print-decl*
10. *var-decl* → *tipo-idt identificador*
11. *tipo-idt* → **int | float | void**
12. *if-decl* → **if** *exp { decl-sequência }* [**else { decl-sequência }**]
13. *atribuição-decl* → **identificador = exp**
14. *while-decl* → **while** *exp { decl-sequência }*
15. *read-decl* → **read identificador**
16. *print-decl* → **print exp**
17. *exp* → *simples-exp [comparação-op simples-exp]*
18. *comparação-op* → **< | > | == | != | <= | >=**
19. *simples-exp* → *termo | termo soma termo*
20. *soma* → **+** | **-**
21. *termo* → *fator | fator mult fator*
22. *mult* → ***** | **/**
23. *fator* → **(exp) | número | identificador**

Pedido:

1. Elaborar o analisador sintático que leia um código fonte no formato .txt.
2. Gerar a recuperação de erros da análise sintática no formato .txt.
3. Gerar a árvore de análise sintática.
4. Gerar a tabela de símbolos.