Lista 1 - Compiladores 2021/2 - UFSJ

Prof. Alexandre Bittencourt Pigozzo.
Valor da lista: 35 pontos.
Data de entrega: **09/11/2021**.
A lista 1 pode ser feita **individualmente ou em dupla. Entregar através do portal didático todos os exercícios, respostas, documentos e códigos compactados em um arquivo com o nome do aluno ou da dupla**.

Exercício 1 (0,5)

Considerando a linguagem Csmall como a linguagem fonte analisada pelo compilador, responda a seguinte questão:
O analisador léxico consegue reconhecer algum erro no código abaixo? Explique destacando os erros léxicos caso existam.

```
main int ( { }
      2 = a ;
}
```

Exercício 2 (0,5)

Considerando a linguagem Csmall como a linguagem fonte analisada pelo compilador, responda a seguinte questão: O analisador léxico consegue reconhecer algum erro no código abaixo? Explique destacando os erros léxicos caso existam.

```
int () {
      int a 2
      int b$;
      float 2a;
}
```

Exercício 3 (1,0)

Com base na gramática abaixo, responda às seguintes questões:

E → T E'
E' → + T E' | - T E' | ^ T E' | * T E' | ε
T → id | num | (E)

Terminais = {+,-,^,*,id,num,(,)}

Obs: O operador ^ é o operador de potência. A relação de precedência da maior para a menor é a seguinte: ^, *, + ou -. O + e - possuem a mesma precedência.

a) Existem conflitos nessa gramática, isto é, conflitos na escolha de uma produção a ser aplicada por um analisador descendente? Justifique sua resposta.
b) Por que a gramática acima não considera a precedência correta de operadores? Justifique.
c) Modifique a gramática acima para que ela passe a considerar a precedência correta de operadores. Caso exista conflitos, elimine os conflitos também. Explique todas as modificações realizadas e mostre a gramática resultante.

Exercício 4 (0,5)

Dada a gramática abaixo, mostre o passo a passo da análise da cadeia id - float_const / integer_const realizada por um autômato de pilha.

E -> T E'
E' -> + T E' | - T E' | ε
T -> F T'
T'-> * F T' | / F T' | ε
F -> id | integer_const | float_const

Terminais = {+,-,*,/,id,integer_const,float_const}

Exercício 5 (1,0)

Com base na gramática abaixo, faça o que se pede:

Construa todas as funções para a implementação do método de descida recursiva. Considere que já foi definida uma função match(tokenesperado) que compara o token atual da entrada com o token esperado passado por parâmetro e avança na entrada em caso de correspondência.

Funcao → function id ( ) ComandoSeq endfunction
ComandoSeq → Comando ; ComandoSeq | ε
Comando → id Comando2
Comando2 → ( ) | = Expr
Expr → [Param_Expr]
Param_Expr → integer_const Param_Expr2
Param_Expr2 → : integer_const Param_Expr2 | ε

Terminais = {function,id,(,),endfunction,;,=,[,],integer_const,:}

Exercício 6 (1,5)

Construa e posicione regras semânticas na gramática abaixo para realizar a **verificação de compatibilidade de tipos** nas expressões da linguagem e nas atribuições. Considere que o tipo do identificador já está armazenado na entrada da tabela de símbolos dele. Considere que um identificador pode ter os seguintes tipos: char, short, int e float. Caso as expressões envolvam o tipo char com int ou float, o compilador deve exibir uma mensagem de erro informando a incompatibilidade. O mesmo é verdadeiro para expressões que tenham short com int ou float como operandos. Expressões envolvem apenas int e float são válidas. As compatibilidades descritas anteriormente também se aplicam às atribuições.

Dica: Ao pensar nas regras semânticas, lembre-se de definir um ou mais atributos que serão utlizados nas regras semânticas. Não esqueça de posicionar todas as regras na gramática.

Assign -> id = E ; AssignList
AssignList -> id = E ; AssignList | ε
E -> T E'
E'-> + T E' | - T E' | ε
T -> F T'
T' -> * F T' | / F T' | ε
F -> (E)
F -> integer_const
F -> float_const
F -> id

Terminais = {+,-,*,/,(,),integer_const,float_const,id,=,;,ε}
integer_const representa uma constante numérica inteira
float_const representa uma constante numérica de ponto flutuante

Exercício 7 (30,0)

A ideia desse exercício é aprender na prática vários conceitos estudados na disciplina. Nesse exercício, vocês irão realizar a implementação das fases de análise léxica, sintática e realizar algumas ações semânticas.
A implementação pode ser feita em qualquer linguagem de programação (ver observação a seguir).

Obs: Recomenda-se que o trabalho seja implementado nas linguagens de programação mais tradicionais e conhecidas como C, C++, Python, Java, Javascript, etc. Caso você queira implementar em uma linguagem não tão conhecida, favor enviar todas as bibliotecas e instruções de instalação para executar o programa no sistema operacional Ubuntu 20.04.

Poderá ser atribuída nota zero ao trabalho caso o professor não consiga executar o trabalho no Ubuntu 20.04.

Com base na gramática abaixo que define uma linguagem similar à linguagem Pascal que chamaremos de linguagem P, faça o que se pede:

Programa -> program id ; Bloco
Bloco -> DeclaracaoSeq begin ComandoSeq end
DeclaracaoSeq -> Declaracao DeclaracaoSeq
DeclaracaoSeq -> ε
Declaracao -> var VarList : Type ;
VarList -> id VarList2
VarList2 -> , id VarList2
VarList2 -> ε
Type -> boolean
Type -> integer
Type -> real
Type -> string
ComandoSeq -> Comando ComandoSeq
ComandoSeq -> ε
Comando -> id := Expr ;
Comando -> if Expr then ComandoSeq end
Comando -> while Expr do ComandoSeq end
Comando -> print Expr ;
Comando -> read id ;
Expr -> Rel ExprOpc
ExprOpc -> OpIgual Rel ExprOpc
ExprOpc -> ε
OpIgual -> ==
OpIgual -> !=
Rel -> Adicao RelOpc
RelOpc -> OpRel Adicao RelOpc
RelOpc -> ε
OpRel -> <
OpRel -> <=
OpRel -> >
OpRel -> >=
Adicao -> Termo AdicaoOpc
AdicaoOpc -> OpAdicao Termo AdicaoOpc
AdicaoOpc -> ε
OpAdicao -> +

OpAdicao -> -
Termo -> Fator TermoOpc
TermoOpc -> OpMult Fator TermoOpc
TermoOpc -> ε
OpMult -> *
OpMult -> /
Fator -> id
Fator -> integer_const
Fator -> real_const
Fator -> TRUE
Fator -> FALSE
Fator -> STRING_LITERAL
Fator -> (Expr)

Obs 1: id é o token para os identificadores. Um identificador, ou seja, o nome de uma variável pode começar com letra e depois ser seguido por letra, número e underline sendo representados pela expressão regular [a-zA-Z]([a-zA-Z0-9_])*.
Obs 2: integer_const é o token que representa os números inteiros [0-9]([0-9])* e real_const é o token para os números de ponto flutuante. Considere que somente são aceitos números de ponto flutuante com dígitos antes e depois do ponto. Ex: 2.4, 534.55. Não são aceitos números na notação científica.
Obs 3: O operador := é o operador de atribuição.

Obs 4: "[a-zA-Z0-9_,;?#]*" é o lexema para as strings literais (cadeias de caracters). Uma string literal começa e termina com aspas duplas. A expressão [a-zA-Z0-9_,;?#]* dentro da string é uma expressão regular que representa uma palavra com zero ou mais combinações de letras minúsculas, letras maiúsculas, números, underline, vírgula, ponto e vírgula, interrogação, cerquilha ou "jogo da velha" e espaço. Exemplos de strings válidas: "sdfk23", "2344", "casa", "Tempo", "SOL", "skdfkm23235FNFOWOF?", "Rua Fulad_2 139, 234 #".

Obs 5: TRUE é o token para o lexema true, isto é, é o token para o valor literal true. FALSE é o token para o lexema false, isto é, é o token para o valor literal false.

Obs 6: STRING_LITERAL é o token para o lexema "[a-zA-Z0-9_,;?#]*" das strings literais.

Obs 7: Na implementação, você pode considerar os nomes dos tokens como sendo formados somente por letras maiúsculas.

a) Construa um autômato finito determinístico (AFD) para reconhecer todos os lexemas da linguagem P.
Os lexemas da linguagem P são os seguintes: **program [a-zA-Z]([a-zA-Z0-9_])* ; begin end**

var : , boolean integer real string := if then while do print read == != < <= > >= + - * / [0-9] ([0-9])* [0-9]([0-9])*.[0-9]([0-9])* true false ( ) "[a-zA-Z0-9_,;?#]*"

b) Implemente um analisador léxico com base no autômato da letra a para reconhecer todos os lexemas da linguagem.
Obs: Você deve definir os tokens para todos os lexemas da linguagem. Em outras palavras, definir um token para cada lexema. **Alguns nomes de tokens já foram definidos como, por exemplo, integer_const para os números inteiros e real_const para números de ponto flutuante**.

c) Construa os conjuntos First e Follow para cada não terminal da gramática.

d) Implemente um analisador sintático utilizando o **método de descida recursiva**. Durante a análise sintática, **todos os identificadores (variáveis) declarados no código, devem ser inseridos na tabela de símbolos juntamente com a informação de tipo**. A estrutura de dados que será utilizada para implementação da tabela de símbolos é escolha de vocês.

e) Com base nas informações da tabela de símbolos, implemente três verificações semânticas durante a análise sintática do código fonte: **1. Verificar se uma variável foi declarada; 2. Verificar se uma variável foi redeclarada; e 3. Verificar a compatibilidade de tipos nas expressões**. Para a tarefa 3, considere o seguinte:

- boolean (true ou false) é incompatível com real e string;
- string é incompatível com boolean (true ou false), integer e real.

Dado um código fonte de entrada escrito na linguagem P, o seu programa deve compilar o código, imprimindo em arquivos de saída (um arquivo para cada saída) as seguintes informações:

- **Vetor de tokens** (que é a saída do léxico): Deverá ser impressa a lista de tokens com as seguintes informações: Lexema reconhecido, token correspondente e número da linha onde o lexema foi reconhecido.
- **Erros léxicos** (caso existam);
- **Erros sintáticos** (caso existam);
- **Todo o conteúdo da tabela de símbolos**;
- E os **resultados das análises semânticas (erros semânticos caso ocorram)**. Por exemplo, imprimir que a variável x não foi declarada na linha 5.

O nome do código fonte deve ser lido do terminal.
Os erros léxicos, sintáticos e semânticos podem ser impressos em um mesmo arquivo de saída com o log de todos os erros.

Além do código fonte, você deve entregar uma documentação com as seguintes informações:

- Tabela com a lista de lexemas e os tokens definidos para cada lexema;
- **O desenho do AFD construído**;
- **Explicação sobre como o analisador léxico foi implementado (lógica para implementar o AFD)**;
- **Explicação sobre como a tabela de símbolos foi implementada**;
- **Explicação sobre como as ações semânticas foram implementadas**;
- **Dificuldades encontradas, o que foi implementado e o que não foi implementado**;
- **Arquivo README explicando como compilar e executar o código**.

Além do código e da documentação acima, você deve entregar os **resultados de 8 testes diferentes**, ou seja, as saídas para 8 códigos fonte de entrada diferentes. É recomendável que todos os comandos e expressões da linguagem estejam "cobertos" em seus testes, isto é, você deve tentar distribuir nos testes os comandos e expressões da linguagem.

Observações finais

Todos os códigos serão analisados por uma ferramenta anti-plágio e pelo comando diff do linux com o objetivo de verificar a similaridade entre eles.