socrative

PL19-aula5

4 Questions

1. Recordando o conceito e a implementação prática/real do módulo de Análise Léxica (ALex), selecione as afirmações verdadeiras abaixo.

0/0 A Um ALex lê integralmente o programa fornecido como entrada e retorna uma sequência com todos os símbolos terminais reconhecidos.

0/0 B Um ALex vai analisando incrementalmente o programa fornecido como entrada e retorna o código de cada símbolo terminal reconhecido.

0/0 C Um ALex reconhece os símbolos terminais do programa fornecido como entrada e retorna o texto do símbolo reconhecido.

0/0 D Um ALex usa um autómato determinista para reconhecer os símbolos terminais no texto de entrada.

0/0 E Para especificar a tarefa de um ALex é vulgar recorrer-se a sistemas de produção <condição-ação> em que o padrão a reconhecer é escrito com base em ERs.

0/0 F Se fornecêssemos ao módulo de reconhecimento uma hash-table com a string de cada símbolo terminal associada ao respetivo código, era possível criar um ALex muito mais eficiente.

2. Analise a especificação Flex abaixo

```
-------------------------------------------------------
%{
#include "y.parser.h"
/* outras Declaracoes C diversas */
%}

%%
[=+\-*/()]              { return yytext[0]; }
[a-zA-Z][a-zA-Z0-9]* { yylval.vals=strdup(yytext); return ID; }
[0-9]+                  { yylval.vali=atoi(yytext); return NUMI; }
[0-9]+'.'[0-9]+         { yylval.valf=atof(yytext); return NUMF; }
.|\n                    { ; }
%%

int yywrap()  { return(1); }
-------------------------------------------------------------------------
```

e indique as afimações verdadeiras a seguir:

0/0 **A** A partir da especificação acima o Flex vai gerar um Analisador Léxico que reconhece 4 símbolos terminais

0/0 **B** O ALex gerado automaticamente a partir da especificação acima reconhece e retorna ao Parser: 7 sinais e 3 símbolos de classe (ou variáveis), mas não reconhece palavras-reservadas.

0/0 **C** O ALex gerado automaticamente a partir da especificação acima reconhece e retorna ao Parser 7 símbolos sinais e 3 palavras-reservadas.

0/0 **D** A especificação acima é inválida porque faz atribuições a varáveis não-declaradas antes de retornar o código dos símbolos.

3. Considere a seguinte GIC escrita em notação do Yacc

```
S : a D C b
  ;
D : d
  | D ',' d
  ;
C :
  |  c ':' C
  ;
```

e então indique as afirmações seguintes verdadeiras.

0/0 **A** Os símbolos **S**, **D**, e **C** são ditos *Não-Terminais*.

0/0 **B** A GIC apresentada tem 3 *Produções*.

0/0 **C** A GIC apresentada tem 5 Produções.

0/0 **D** A GIC apresentada está incompleta porque há uma produção sem símbolos no *Lado Direito*.

0/0 **E** A frase

a d b

pertence à Linguagem gerada pela GIC apresentada.

4. Atente na Gramática (GIC) abaixo, escrita em notação Yacc

~~~~~~

```
%%
Par  :  '(' Par ')'  Par
     |
     ;
%%
int yyerror(char *s){
       printf("Erro sintático...\n");
       return 0;
}
int main(){
      yyparse();
      return 0;
}
```

~~~~~~

e então diga quais as afirmações abaixo são verdadeiras:

.

0/0 **A** A frase

(())()

pertence à linguagem especificada por esta GIC.

0/0 **B** A frase

()(())()

pertence à linguagem especificada por esta GIC.

0/0 **C** A frase

(( )( ))

pertence à linguagem especificada por esta GIC.

0/0 **D** Apesar da dupla recursividade, evidente na GIC acima, o Yacc aceita a gramática dada.

0/0 **E** O Yacc dará erro ao processar a GIC acima por não existirem Tokens.