socrative

PLC21-Exe4

6 Questions

1. Recorde a Máquina de Stack Virtual que estudou, VM, e a sua linguagem Assembly. Indique então qual das seguintes afirmações é verdadeira:

10/17 **A** se num programa fonte forem declaradas 4 variáveis do tipo inteiro, é possível usar a instrução Assembly **PUSHN 4** para reservar memória global para essas variáveis.

0/17 **B** se num programa forem declaradas 5 variáveis do tipo inteiro e dois arrays de inteiros de 10 e 20 posições, não é possível usar a instrução Assembly **PUSHN 35** para reservar memória global para essas variáveis.

3/17 **C** qualquer programa em Assembly tem de começar obrigatoriamente com a instrução **START**.

4/17 **D** para saltar forçosamente para uma determinada instrução, sem ser a instrução seguinte do programa, é obrigatório saber o índice dessa instrução destino na memória de programa.

2. Considere o seguinte programa Assembly da VM:

```
PUSHI 10
PUSHI 2
PUSHI 10
beg: START
JUMP main
f1: NOP
READ
ATOI
STOREG 0
RETURN
main: NOP
PUSHG 0
PUSHG 2
SUB
JZ beg
STOP
```

Selecione, então, das alíneas abaixo a afirmação verdadeira:

1/18 **A** o código entre '**f1:**' e '**main:**' é sempre executado ao arrancar com o programa.

4/18 **B** o código que começa em '**f1:**' não seria executado mesmo que a instrução '**SUB**' fosse substituída pela instrução '**JUMP f1**'.

9/18 **C** o programa apresentado acima nunca termina.

4/18 **D** o programa apresentado acima termina sempre ao fim da primeira execução.

3. Considere o seguinte excerto de um ficheiro Python (escrito com base no módulo '**ply.yacc**') que implementa um processador o qual reconhece uma sequência de intervalos fechados de números inteiros e escreve no fim um valor *enigma:*

```
def p_sequencia(p):
    "sequencia : intervalos"
    print(p[1])

def p_intervalos_intervalo(p):
    "intervalos : intervalo"
    p[0] = p[1]

def p_intervalos_intervalos(p):
    "intervalos : intervalos intervalo"
    p[0] = p[2] if p[1] < p[2] else p[1]

def p_intervalo(p):
    "intervalo : '[' NUM ',' NUM ']'"
    p[0] = p[4] - p[2] if p[2] <= p[4] else -1
```

Selecione então a afirmação abaixo que é verdadeira:

12/18 **A** se o texto de entrada for '**[3,1] [1,4] [2,3**]' o processador deverá escrever **3.**

1/18 **B** no caso do texto de entrada conter pelo menos um intervalo em que o limite inferior é maior que o limite superior, o processador escreverá sempre **-1**.

1/18 **C** O processador imprime a largura do menor intervalo da sequência, caso haja pelo menos um intervalo válido.

4/18 **D** se o texto de entrada for '**[3,3] [4,1] [5,2]**' o processador deverá escrever **3.**

4. Recorde o que aprendeu sobre estratégias para fazer o *parsing (*ou análise sintática) de frases de uma linguagem.
Nesse contexto, analise a veracidade das seguintes afirmações:

12/17 **A** um *parser* **Top-Down** é um *parser preditivo* porque olha para a GIC e impõe o próximo símbolo a reconhecer.

3/17 **B** um *parser* **Bottom-Up** é um *parser preditivo* porque olha para a GIC e impõe o próximo símbolo a reconhecer.

0/17 **C** um *parser* **Top-Down** constrói a Árvore de Derivação de uma frase correta partindo das folhas e juntando-as até chegar à raiz.

2/17 **D** um *parser* diz-se **Bottom-Up** se constrói a Árvore de Derivação de uma frase correta partindo da raíz e derivando até encontrar folhas válidas.

5. Considere a seguinte GIC, sabendo que **T** = {**a,f,di,da,i,p**}:

Frase => a Corpo f
Corpo => Ds Is
Ds => di
Ds => da
Is => i Resto
Resto => €
| p i Resto

Selecione, então, das alíneas abaixo a afirmação verdadeira:

9/16 **A** um *parser* **Descendente**, ou **TD**, consegue reconhecer as frases geradas por esta GIC.

2/16 **B** dado que existe uma produção em que um símbolo deriva em *vazio*, não é possível usar um *parser* **Top-Down** para reconhecer as frases da linguagem gerada pela GIC.

3/16 **C** o facto do símbolo **Ds** ter mais do que uma alternativa para derivar impede o uso de um *parser* **Top-Down**

2/16 **D** uma vez que existe recursividade à direita nesta GIC, não será possível reconhecer as frases por ela geradas com um *parser* **Bottom-Up**.

6. Considere a seguinte GIC, sabendo que **T** = {**a,f,di,da,i,p**}:

Frase => a Corpo f
Corpo => Ds Is
Ds => di
Ds => da
Is => i Resto
Resto => €
| p i Resto

Suponha ainda que a variável "**fonte**" é um array de símbolos terminais que corresponde à frase de entrada a analisar após ter sido processada pelo analisador léxico
e que o índice "**i**" seleciona o próximo símbolo dessa frase a ser reconhecido (sendo que "**i**" avança uma posição sempre que um terminal é aceite).
Diga, então, se a afirmação abaixo é verdadeira:
*As duas funções cujos algoritmos se mostram abaixo servem para Reconhecer os símbolos não-terminais "**Frase**" e "**Corpo**" usando a estratégia **Top-Down***

```
recFrase( fonte[i] )
  se ( fonte[i] == a )
  entao recT( a, fonte[i] );
        recCorpo( fonte[i] );
        recT( f, fonte[i] )
  senao erro()

recCorpo( fonte[i] )
  se (( fonte[i] == di ) ou ( fonte[i] == da ))
  entao recDs( fonte[i] );
       recIs( fonte[i] )
  senao erro()
```

11/17 T True

6/17 F False