Mini-Teste 2 (2015/06/29)

Nome: _____ **Número:** _____

Duração: 1h45 **Versão B**

Prova com consulta limitada a 1 folha (2 páginas) A4 manuscrita.
Não são permitidos meios eletrónicos (computador, telemóvel, …).
Tentativas de fraude conduzem à anulação da prova para todos os intervenientes.

Responda aos grupos I e II numa folha separada!

Coloque o seu nome completo e a versão do exame em todas as folhas!

Grupo I: [3,5 Val] Análise Semântica e Representações Intermédias

Considere o excerto de código à direita.

```
1.  int func(int var) {
2.      int b = 2;
3.      globB[var] += b;
4.      globC[var] *= b;
5.      if( var < globA)
6.          return globB[var];
7.      else
8.          return globC[var];
9.  }
```

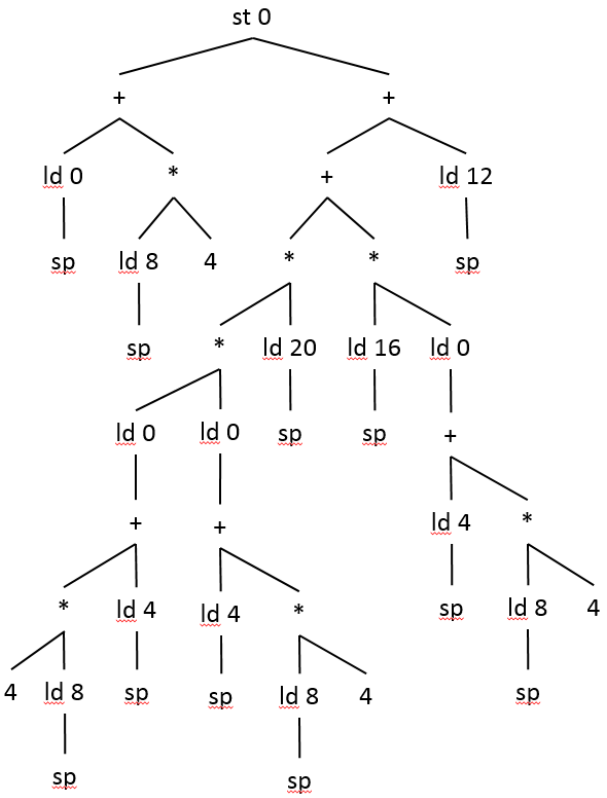
a) Identifique que tipos de validações semânticas podem ser aplicadas em cada linha deste excerto de código.

b) Desenhe árvores de representação intermédia de alto e de baixo nível para a instrução da linha 4 do excerto de código, de acordo com a representação estudada nas aulas (assuma que os inteiros ocupam 4 bytes e que as variáveis se encontram armazenadas nas posições indicadas na tabela abaixo).

var	int	SP
b	int	SP + 4
globA	int	DP
globB	int[]	DP + 4
globC	int[]	DP + 8

Grupo II: [3 Val] Seleção de Instruções

Considere a representação intermédia ao lado, em que ***sp*** representa o registo com o endereço da pilha (*Stack Pointer*). Assuma que os inteiros ocupam 4 bytes.



a) Indique um possível trecho de código de uma linguagem de programação em alto-nível (C ou Java) que esteja na origem desta representação.

b) Considere o conjunto de instruções apresentado abaixo. Utilizando o algoritmo *Maximum Munch* obtenha uma cobertura para a árvore de representação intermédia apresentada ao lado, e apresente uma sequência de instruções para a sua execução. Nota: instruções ld (LOAD) correspondem a acessos de leitura à memória (MEM) e instruções st (STORE) correspondem a acessos de escrita na memória (MOVE).

Name	*Effect*	*Trees*
—	r_i	TEMP
ADD	$r_i \leftarrow r_j + r_k$	+
MUL	$r_i \leftarrow r_j \times r_k$	*
SUB	$r_i \leftarrow r_j - r_k$	-
DIV	$r_i \leftarrow r_j / r_k$	/
ADDI	$r_i \leftarrow r_j + c$	+ CONST; + CONST; CONST
SUBI	$r_i \leftarrow r_j - c$	- CONST
LOAD	$r_i \leftarrow M[r_j + c]$	MEM + CONST; MEM + CONST; MEM CONST; MEM
STORE	$M[r_j + c] \leftarrow r_i$	MOVE MEM + CONST CONST; MOVE MEM + CONST; MOVE MEM CONST; MOVE MEM
MOVEM	$M[r_j] \leftarrow M[r_i]$	MOVE MEM MEM

Grupo III: [3,5 Val] Análise de fluxo de dados

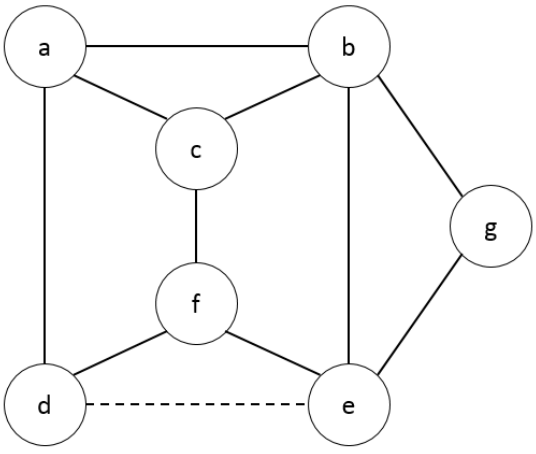
a) Obtenha o CFG (*Control Flow Graph* – Grafo de Fluxo de Controlo) para o excerto de código apresentado ao lado.

b) Realize a análise de tempo de vida (em sentido inverso - *backward*) para o CFG obtido na alínea anterior. Apresente as iterações necessárias para a análise, apresentando os conjuntos *def*, *use*, *in* e *out*. Apresente o grafo de interferência resultante da análise de tempo de vida.

```
1.  int func(int par) {
2.      int bop = par%2;
3.      int liv = 0;
4.      while(par > 1) {
5.          if(bop == 0)
6.              par = par / 2;
7.          else
8.              par = par + 1;
9.          bop = par % 2;
10.         liv ++;
11.     }
12.     if(liv > 10)
13.         return liv;
14.     else
15.         return par;
16.     }
```

Grupo IV: [3 Val] Alocação de Registos

Considere o grafo de interferência de registos apresentado abaixo, em que a linha tracejada indica uma relação de movimentação (*move-related*).



a) Baseado no grafo de interferência de registos apresentado ao lado, realize a atribuição de registos usando coloração de grafos. Assuma a existência de 3 registos disponíveis ($a, $b e $c). Apresente o estado da pilha após simplificação do grafo, e a atribuição final de registos a cada variável. Caso tenha de realizar *spilling*, indique o critério para seleção da variável. Apresente o resultado da primeira coloração.

Grupo V: [2] Otimizações

Considere o excerto de código apresentado ao lado.

a) Indique uma sequência possível de otimizações possíveis de realizar sobre o código apresentado. Indique todos os passos de otimização aplicados, apresentando o novo código após cada passo de otimização.

```
1. int a = 1;
2. int v = 2;
3. int x = 3;
4. int t = 2;
5. int m = v * t;
6. x = x + m;
7. m = t * t;
8. m = m * a;
9. x = x + m;
```

Grupo VI: [5 Val] Compiladores (Miscelânea)

a) Explique sucintamente o papel das tabelas de símbolos na fase de análise semântica.

b) Explique sucintamente como é realizada a validação semântica de uma chamada a uma função (validação de número e tipo dos argumentos e do tipo de retorno).

Indique, **justificando sucintamente**, se cada uma das seguintes afirmações é Verdadeira ou Falsa. (Resposta errada = desconto de 50% da cotação da alínea)

c) Numa linguagem como Java, a fase de análise semântica requer apenas uma passagem pela árvore sintática do programa.

d) No âmbito do problema de seleção de instruções utilizando programação dinâmica, a melhor solução é sempre a solução com um menor número de instruções.

e) A realização da análise de tempo de vida de variáveis em sentido inverso (*backward*) permite normalmente poupar algumas iterações em comparação com a realização da mesma análise em sentido direto (*forward*).

f) É impossível obter uma **k**-coloração válida (isto é, colorir o grafo usando **k** cores) a partir de um grafo de interferência de registos em que todos os nós têm grau igual a **k**.

g) A realização de operações de otimização no processo de compilação de uma linguagem é computacionalmente pesada e na prática não conduz a diferenças significativas em relação a uma compilação não otimizada.

Boa sorte!