COMPILADORES

Prova 2 – Prof. Marcus Ramos – 25 de abril de 2013

Questão 1 (1,6 ponto):

Sobre a fase de análise de contexto:

- Explique o funcionamento da subfase de identificação, e mostre como ela pode garantir que as regras de escopo estejam sendo obedecidas pelo programa-fonte;
- Explique o funcionamento da subfase de verificação de tipos, e mostre como ela pode garantir que as regras de tipo da linguagem estejam sendo obedecidas pelo programa-fonte;
- Explique o que muda na subfase de identificação quando são consideradas linguagens com escopos dinâmicos;
- Explique o que muda na subfase de verificação de tipos quando são consideradas linguagens com tipos dinâmicos.

Questão 2 (1,6 ponto):

Considere a seguinte trecho de programa Pascal:

```
var a: array [1..10] of record
                m: integer;
                n: array [-20..20] of record
                          x: real;
                          y: boolean
                          end
                end
```

Considere:

- base (a)=1000;
- size(boolean)=1;
- size(integer)=2;
- size(real)=4.

Determine:

- size(a);
- address(a[i].m);
- address(a[i].n[j]);
- address(a[i].n[j].y).

```
size (a)=(10-1+1)*(2+(20-(-20)+1)*(4+1)+2)=2.070 bytes.
address(a[i].m)=1000+(i-1)*207.
address(a[i].n[j])=1000+(i-1)*207+2+(j+20)*5.
address(a[i].n[j].y)=1000+(i-1)*207+2+(j+20)*5+4.
```

Questão 3 (1,6 ponto):

Sobre alocação de memória:

- Explique quando é usada, como funciona e como são atribuídos endereços para variáveis com alocação estática;
- Explique quando é usada, como funciona e como são atribuídos endereços para variáveis com alocação automática;
- Explique quando é usada, como funciona e como são atribuídos endereços para variáveis com alocação dinâmica;
- Descreva estratégias para reduzir a fragmentação do *heap* e minimizar o insucesso na alocação de novas variáveis.

Questão 4 (1,8 ponto):

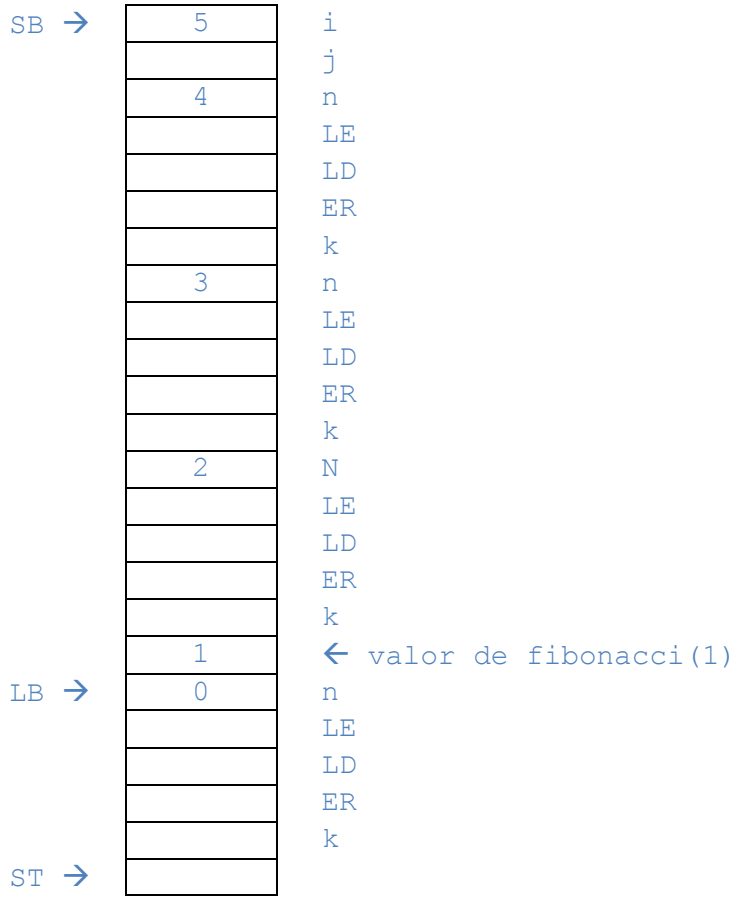
- O que é e para o que serve um "frame"?
- O que é, onde fica armazenado e para que serve o link estático de um frame?
- O que é, onde fica armazenado e para que serve o link dinâmico de um frame?
- O que é, onde fica armazenado e para que serve o endereço de retorno de um frame?
- Como e onde são representadas as variáveis locais de um bloco?
- Como e onde são representados os parâmetros de um bloco?

Questão 5 (3,4 pontos):

Considere o programa em C abaixo.

```
#include <stdio.h>
int fibonacci (int n) {
    int k;
    if (n==0) k=0;
        else if (n==1) k=1;
        else k=fibonacci (n-1)+fibonacci(n-2);
    return k;
}
int main () {
    int i,j;
    scanf ("%d",&i);
    while (i>0) {
        j=fibonacci(i-1);
        printf ("%d\n\n",j);
        scanf ("%d",&i);
    }
}
```

- Suponha que o usuário digite na entrada o valor 5. Mostre, no início da execução da função "fibonacci", quando o parâmetro "n" assume o valor 0 pela primeira vez, a situação completa da pilha de execução, com alocação de variáveis, parâmetros, ponteiros, links etc, todos com o seus respectivos valores;
- Mostre o código gerado para o mesmo, considerando a linguagem objeto TAM e os padrões de código vistos em sala de aula. Todas as variáveis e parâmetros deverão ser referenciados através dos seus respectivos endereços, na forma deslocamento e registrador. Considere que ponteiros, variáveis e parâmetros ocupam uma única posição de memória cada.




```
main:

      PUSH              2
      LOADA             0[SB]
      CALL              SCANF
R2:   LOAD              0[SB]
      LOADL             0
      CALL              GT
      JUMPIF     (1)    R1
      LOAD              0[SB]
      LOADL             1
      CALL              SUB
      CALL              FIBONACCI
      STORE             1[SB]
      LOAD              1[SB]
      CALL              PRINTF
      LOADA             0[SB]
      CALL              SCANF
      JUMP              R2
R1:   HALT

fibonacci:

      PUSH              1
      LOAD              -1[LB]
      LOADL             0
      CALL              EQUAL
```

```
      JUMPIF     (0)   R2
      LOADL            0
      STORE            3[LB]
      JUMP             R3
R2:   LOAD             -1[LB]
      LOADL            1
      CALL             EQUAL
      JUMPIF     (0)   R4
      LOADL            1
      STORE            3[LB]
      JUMP             R5
R4:   LOAD             -1[LB]
      LOADL            1
      CALL             SUB
      CALL             FIBONACCI
      LOAD             -1[LB]
      LOADL            2
      CALL             SUB
      CALL             FIBONACCI
      CALL             ADD
      STORE            3[LB]
R5:
R3:   LOAD             3[LB]
      RETURN (1)       1
```