Exercício 5 de Compiladores

Nome: Samuel Evangelista de Aquino Junior
Matricula: 397618

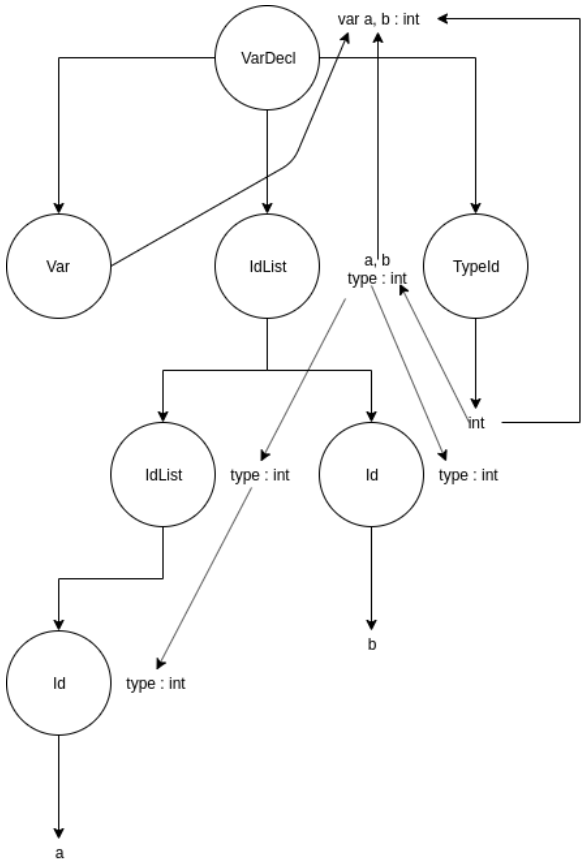
1.

a)

VarDecl →	var IdList : TypeId	{IdList.type = TypeId}
IdList →	IdList*, Id	{Id.type = IdList.type IdList*.type = IdList.type }
|	Id	{Id.type = IdList.type}

Agora vamos mostrar a derivação mais a direita de var a, b : int e mostrar a árvore e o grafo de dependência.

VarDecl → var IdList : TypeId
→ var IdList : int
→ var IdList, Id : int
→ var IdList, b : int
→ var Id, b : int
→ var a, b : int



b)

```
bárvore →( number bárvore esq bárvore dir )
                                        {
                                        /* Assumiremos que não está ordenado, começando da raiz */
                                        bárvore.ord = false
                                        bárvore.val = number

                                        /* Caso ele não tenha filhos, já está ordenado */
                                        if(bárvore esq == nil && bárvore dir == nil)
                                                bárvore.ord = true

                                        /* Caso tenha dois filhos, faremos a compração dos dois filhos */
                                        else if(bárvore esq != nil && bárvore dir != nil)
                                                /* Verificar a ordem dos filhos desse nó */
                                                if(bárvore esq .ord && bárvore dir .ord)
                                                        if(bárvore esq .val <= number && bárvore dir .val
                                                        >=number)
                                                                bárvore.ord = true

                                        /* Caso so tenha o filho esquerdo, vamos verificar a ordem */
                                        else if(bárvore esq != nil)
                                                if(bárvore esq .ord && bárvore esq .val <= number)
                                                        bárvore.ord = true

                                        else /* Caso so tenha o filho direito, vamos verificar a ordem */
                                                if(árvore dir .ord && bárvore dir .val >= number)
                                                        bárvore.ord = true
                                        }

|       nil                           { bárvore.ord = true }
```

2)

Fibn: -> supondo que n está em r0

```
loadI 1, r1 //carregar 1 para o r1
loadI @x, r2 //carregar o endereço de x para r2
store r1, r2 // colocar na memoria do endereço de x o valor 1
loadI 1, r3 //carregar 1 para o r3
loadI @y, r4 //carregar o endereço de y para r4
store r3, r4 //colocar na memoria do endereço de y o valor 1
loadI 1, r5 //carregar 1 para o r5
loadI @z, r6 //carregar o endereço de z para r6
store r5, r6 //colocar na memoria do endereço de z o valor 1
load 1, r7 //colocar 1 no r7 para fazer a comparação do while(n > 1)
COMP:
    cmp_GT r0, r7, r8 //comparação n > 1, true → r8 = true, false → r8 = false
    cbr r8, FIB, CONT //pular para a label de executar o FIB enquanto r8 = true
FIB:
    loadI @x, r9 //carregar o endereço de x em r9
    load r9, r10 //carregar o valor do endereço de x em r10
    loadI @y, r11 //carregar o endereço de y em r11
    load r11, r12 //carregar o valor do endereço de y em r12
    add r10, r12, r13 //somar x + y e guardar em r13
    loadI @z, r14 //carregar o endereço de z em r14
    store r13, r14 //colocar a soma de x+y no endereço de z
    store r12, r9 //colocar o valor de y no endereço de x
    load r14, r15 //carrego o valor do endereço de z para r15
    store r15, r11 //coloco o valor de z no endereço de y
    subI r0, 1, r0 //decremeto o valor de n que está em r0
    jmpI COMP //faço jump para a comparação novamente
CONT:
.....…  //o espaço de memoria de z tem o resultado do fib(n)
```