UNISINOS - UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6) – Curso: Informática

LABORATÓRIO II – AULA 09

Disciplina: Linguagem de Programação PASCAL **E-mail:** ***osorio@exatas.unisinos.br***
Professor responsável: ***Fernando Santos Osório*** **Web:**
Semestre: 99/2 ***http://www.inf.unisinos.br/~osorio/lab2.html***
Horário: 21 e 41 **Xerox : *Pasta 54* – LAB. II (Xerox do C6/6)**

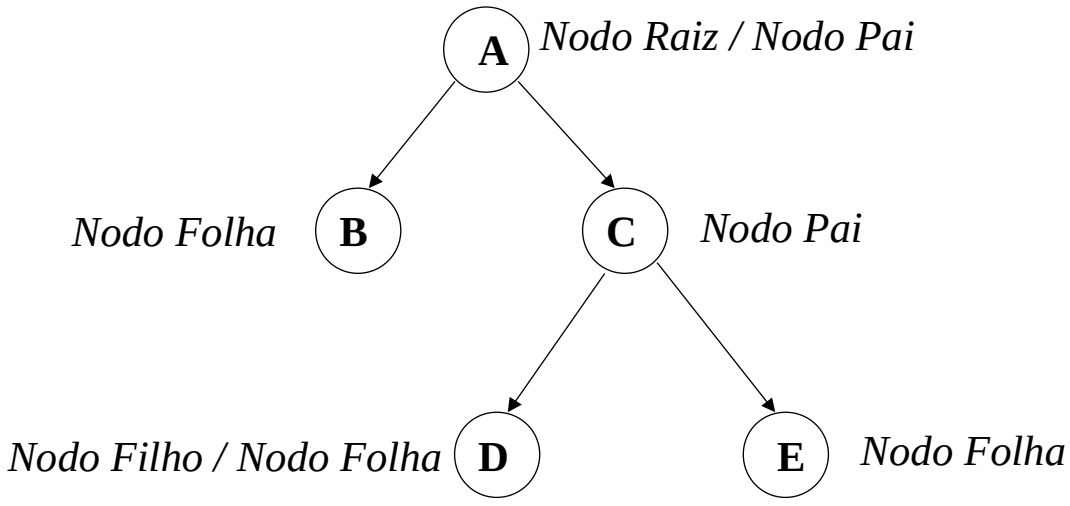
ÁRVORES BINÁRIAS
Alocação Dinâmica de Memória

As árvores são estruturas de dados criadas usualmente através do uso de alocação dinâmica de memória, baseadas em listas encadeadas, que possuem um nodo superior (raiz / pai), apontando para os seus nodos filhos (folhas / filho). Por sua vez cada nodo pai pode possuir nodos filhos, e assim chegamos a definição "recursiva" de uma árvore: um nodo pai aponta para nodos filhos, onde estes nodos filhos também podem ser nodos pais.

Uma árvore binária é um tipo específico de árvore que possui apenas 2 nodos filhos ligados a cada nodo pai: o nodo da esquerda e o nodo da direita. As árvores binárias permitem uma busca mais eficiente quando os dados são inseridos de forma ordenada, podendo ser percorridas de três maneiras:

- Percorrer de modo prefixado: Pré-Ordem (VED = Visita/Esquerda/Direita);
- Percorrer de modo infixado : Em Ordem (EVD = Esquerda/Visita/Direita);
- Percorrer de modo pósfixado: Pós-Ordem (EDV = Esquerda/Direita/Visita);

A figura abaixo mostra um exemplo de uma árvore binária:



Usualmente teremos a seguinte estrutura de dados para uma ***árvore binária***:

```
Type
    Tipo_Dado          =  Integer;
    Ptr_Nodo_AB        =  ^Nodo_Arv_Bin;
    Nodo_Arv_Bin       =  Record
                             Dado: Tipo_Dado;
                             Pai: Ptr_Nodo_AB;
                             ArvEsq, ArvDir : Ptr_Nodo_AB;
                          End;
Var
    Arvore_Binaria: Ptr_Nodo_AB;
    Novo_Nodo     : Ptr_Nodo_AB;

{ Criando uma árvore binária com 3 nodos apenas...                    }
{ Note que em uma árvore binária os nodos podem ser inseridos         }
{ de maneira ordenada, ou seja:                                       }
{ - Os nodos a esquerda de um nodo pai são sempre menores que ele     }
{ - Os nodos a direita de um nodo pai são sempre maiores que ele      }

Begin
    New (Novo_Nodo);              { Alocação dinâmica: cria o nodo raiz (pai)     }
    Novo_Nodo^.Dado := 10;        {  Coloca o dado no nodo que foi alocado        }
    Novo_Nodo^.Pai := NIL         { O nodo raiz não tem pai (sem nodos acima)     }
    Novo_Nodo^.ArvEsq := NIL ;    {  Inicialmente o nodo não tem filho (esq.)     }
    Novo_Nodo^.ArvDir := NIL;     {  e também não filho (direita)                 }
    Arvore_Binaria:= Novo_Nodo;{ Arvore_Binaria aponta para a Raiz da árvore}

    New(Novo_Nodo);               { Alocação dinâmica: cria o nodo filho esq.     }
    Novo_Nodo^.Dado := 5;         { Coloca o dado no nodo que foi alocado         }
    Novo_Nodo^.Pai :=Arvore_Binaria;      { O pai do novo nodo é o nodo raiz      }
    Novo_Nodo^.ArvEsq := NIL;     {  Inicialmente o nodo não tem filho (esq.)     }
    Novo_Nodo^.ArvDir := NIL;     {  e também não filho (direita)                 }
    Arvore_Binaria^.ArvEsq :=Novo_Nodo; { O filho esq. da raiz é o novo nodo      }

    New(Novo_Nodo);               { Alocação dinâmica: cria o nodo filho dir.     }
    Novo_Nodo^.Dado := 15;        { Coloca o dado no nodo que foi alocado         }
    Novo_Nodo^.Pai :=Arvore_Binaria;      { O pai do novo nodo é o nodo raiz      }
    Novo_Nodo^.ArvEsq := NIL;     {  Inicialmente o nodo não tem filho (esq.)     }
    Novo_Nodo^.ArvDir := NIL;     {  e também não filho (direita)                 }
    Arvore_Binaria^.ArvDir :=Novo_Nodo; { O filho dir. da raiz é o novo nodo      }
End.
```

- As árvores binárias ordenadas permitem que se faça a chamada pesquisa binária, pois os nodos estão ordenados, onde sempre a esquerda de um nodo se encontram valores menores que o valor deste nodo, e a sua direita sempre temos valores maiores que o valor deste nodo. Note também que a árvore acima foi criada usando uma lista duplamente encadeada (pai aponta para os filhos – esq./dir., e filho aponta para o pai). As listas duplamente encadeadas permitem que se percorra a lista/árvore nos dois sentidos, o que pode ser bastante interessante de acordo com o problema a ser tratado... entretanto, este tipo de encadeamento duplo pode não ser necessário em muitas situações. Nas rotinas que serão implementadas a seguir, podemos percorrer toda a árvore de forma recursiva, sem precisar usar o ponteiro de "nodo pai".

EXERCÍCIOS – *Árvores Binárias*

1. Faça um programa para a manipulação de árvores binárias usando listas encadeadas com alocação dinâmica na memória do computador. Crie as seguintes rotinas genéricas de manipulação de dados:

   Definições elementares:

```
Type
    Tipo_Dado       = Integer;
    Ptr_Nodo_AB     = ^Nodo_Arv_Bin;
    Nodo_Arv_Bin    = Record
                        Dado: Tipo_Dado;
                        Pai: Ptr_Nodo_AB;
                        ArvEsq, ArvDir : Ptr_Nodo_AB;
                      End;
```

   Rotinas:

```
{ Inicializa a árvore binária – Prepara para inserir dados }
Procedure Inicializa_ArvBin (Var Raiz: Ptr_Nodo_AB);

{ Insere um dado de modo ordenado na árvore binária }
Procedure Insere_ArvBin (Var Raiz: Ptr_Nodo_AB; Dado: Tipo_Dado);

{ Exibe o conteúdo (dados) da árvore em ordem infixada }
Procedure Exibe_AB_Infixado (Raiz: Ptr_Nodo_AB);

{ Exibe o conteúdo (dados) da árvore em ordem prefixada }
Procedure Exibe_AB_Prefixado (Raiz: Ptr_Nodo_AB);

{ Exibe o conteúdo (dados) da árvore em ordem pósfixada }
Procedure Exibe_AB_Posfixado (Raiz: Ptr_Nodo_AB);

{ Apaga toda a árvore binária, liberando a memória ocupada }
Procedure Apaga_ArvBin (Var Raiz: Ptr_Nodo_AB);

{ Procura um dado na árvore e retorna um ponteiro para este dado (ou NIL se não achar) }
Function   Pesquisa_ArvBin (Raiz: Ptr_Nodo_AB; Dado: Tipo_Dado): Ptr_Nodo_AB;

{ Acha o menor valor armazenado na árvore, retornando o valor e o ponteiro para o nodo }
Function   Menor_ArvBin (Raiz: Ptr_Nodo_AB; Var Dado: Tipo_Dado): Ptr_Nodo_AB;

{ Acha o maior valor armazenado na árvore, retornando o valor e o ponteiro para o nodo }
Function   Maior_ArvBin (Raiz: Ptr_Nodo_AB; Var Dado: Tipo_Dado): Ptr_Nodo_AB;

{ Remove o nodo cujo ponteiro foi passado como parâmetro }
Procedure Remove_AB_Nodo (Var Nodo: Ptr_Nodo_AB);
```

```
{ Remove o nodo que contem o valor indicado como parâmetro, indicando se conseguiu }
Function   Remove_AB_Dado (Var Raiz: Ptr_Nodo_AB; Dado: Tipo_Dado): Boolean;

{ Conta quantos nodos possui a árvore }
Function   Conta_ArvBin (Raiz: Ptr_nodo_AB): Integer;

{ Acha o nodo cujo dado é o seguinte em relação ao dado contido no nodo apontado }
Function Sucessor_ArvBin (Nodo: Ptr_Nodo_AB): Ptr_Nodo_AB;

{ Acha o nodo cujo dado é o anterior em relação ao dado contido no nodo apontado }
Function Predecessor_ArvBin (Nodo: Ptr_Nodo_AB): Ptr_Nodo_AB;
```

2. Faça um programa para a manipulação de árvores binárias usando as rotinas genéricas de manipulação de dados descritas no exercício anterior. Leia um arquivo texto com uma seqüência qualquer de números, e gere uma árvore binária ordenada. A seguir, gere um arquivo texto contendo a seqüência ordenada de números que foram armazenados na árvore.