UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CIÊNCIAS EXATAS E TECNOLÓGICAS – Curso: Informática / Ciência da Computação

Programação II

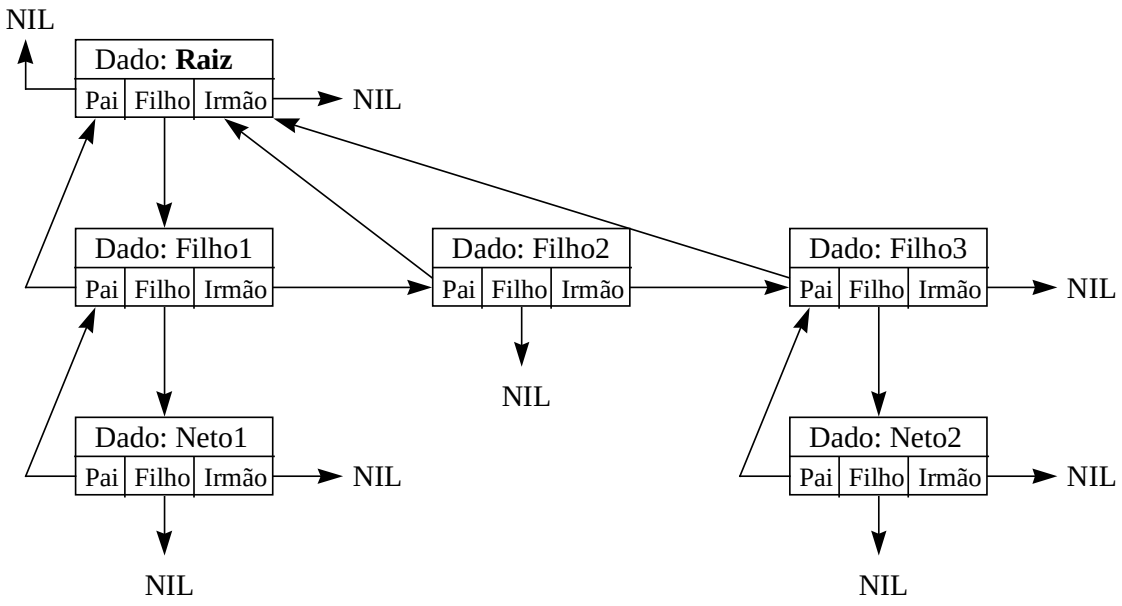
Disciplina: Linguagem de Programação PASCAL **E-mail:** ***osorio@exatas.unisinos.br***
Professor responsável: ***Fernando Santos Osório*** **Web:**
Semestre: 2004/2 ***http://www.inf.unisinos.br/~osorio/prog2.html***
Horário: 63 **Xerox : *Pasta 54* – PROG. II (Xerox do "Alemão")**

ÁRVORES GENÉRICAS
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 genérica é um tipo especial de árvore onde podemos ter um número variável de nodos filhos associados a um nodo pai. Um exemplo de implementação de uma árvore genérica pode ser dado por uma estrutura onde cada nodo possui um ponteiro para o seu nodo pai, um ponteiro para o seu nodo filho, e um ponteiro para o seu nodo irmão. Assim sendo, cada nodo pode possuir um número ilimitado de filhos, pois o seu nodo filho aponta para uma lista ilimitada de nodos no mesmo nível na hierarquia da árvores (irmãos). Além disto, continuamos tendo uma estrutura em árvore, pois cada nodo possui seu nodo pai e seu(s) nodo filho(s).

A figura abaixo mostra um exemplo de uma árvore genérica:



Podemos ter a seguinte estrutura de dados para uma ***árvore genérica***:

```
Type
    TArvGen_Dado          =  Integer;
    Ptr_ArvGen_Nodo       =  ^ArvGen_Nodo;
    ArvGen_Nodo       =  Record
                            Dado: TArvGen_Dado;
                            Pai, Filho, Irmao:  Ptr_ArvGen_Nodo;
                         End;
Var
    Arvore_Generica: Ptr_ArvGen_Nodo;
    Novo_Nodo      : Ptr_ArvGen_Nodo;
    Ultimo_Filho   : Ptr_ArvGen_Nodo;

{ Criando uma árvore genérica com 4 nodos apenas...  divididos em 2 níveis      }
{  2 níveis:  1º. nível – nodo raiz/pai, 2º. nível – 3 nodos filhos              }
{ Usando rotinas não genéricas...    }

Begin
    New (Novo_Nodo);              { Alocação dinâmica: cria o nodo raiz (pai)     }
    Novo_Nodo^.Dado := 1;         {  Coloca o dado no nodo que foi alocado       }
    Novo_Nodo^.Pai := NIL;        { O nodo raiz não tem pai (sem nodos acima)  }
    Novo_Nodo^.Irmao := NIL;      { Não possui nodos irmãos no mesmo nível    }
    Novo_Nodo^.Filho := NIL;      { Inicialmente não possui nodos filhos        }
    Arvore_Generica := Novo_Nodo;  { Arvore_generica aponta para a Raiz      }

    New(Novo_Nodo);               { Alocação dinâmica: cria o nodo filho1       }
    Novo_Nodo^.Dado := 11;        { Coloca o dado no nodo que foi alocado       }
    Novo_Nodo^.Pai :=Arvore_Generica;     { O pai do novo nodo é o nodo raiz  }
    Novo_Nodo^.Filho := NIL;      {  Inicialmente o nodo não tem filho          }
    Novo_Nodo^.Irmao := NIL;      {  e também não tem irmão                     }
    Arvore_Generica^.Filho := Novo_Nodo;  { O filho da raiz é o novo nodo       }
    Ultimo_Filho := Novo_Nodo;    { Indica que este foi o último filho inserido   }

    New(Novo_Nodo);               { Alocação dinâmica: cria o filho2 (irmão)    }
    Novo_Nodo^.Dado := 12;        { Coloca o dado no nodo que foi alocado       }
    Novo_Nodo^.Pai :=Arvore_Generica;     { O pai do novo nodo é o nodo raiz  }
    Novo_Nodo^.Filho := NIL;      {  Inicialmente o nodo não tem filho          }
    Novo_Nodo^.Irmao := NIL;      {  e também não tem irmão (mais moço)         }
    Ultimo_Filho^.Irmao := Novo_Nodo;     { O irmão aponta p/ novo irmão      }
    Ultimo_Filho := Novo_Nodo;    { Indica que este foi o último filho inserido   }

    New(Novo_Nodo);               { Alocação dinâmica: cria o filho2 (irmão)    }
    Novo_Nodo^.Dado := 13;        { Coloca o dado no nodo que foi alocado       }
    Novo_Nodo^.Pai :=Arvore_Generica;     { O pai do novo nodo é o nodo raiz  }
    Novo_Nodo^.Filho := NIL;      {  Inicialmente o nodo não tem filho          }
    Novo_Nodo^.Irmao := NIL;      {  e também não tem irmão (mais moço)         }
    Ultimo_Filho^.Irmao := Novo_Nodo;     { O irmão aponta p/ novo irmão      }
    Ultimo_Filho := Novo_Nodo;    { Indica que este foi o último filho inserido   }

End.
```

Como exemplos de aplicação das árvores genéricas podemos imaginar seu uso para representar a hierarquia de uma "árvore genealógica", armazenando a relação de nível entre pais e filhos (grau de parentesco: avô, pai, filho, neto, etc), bem como a relação de vizinhança entre os diversos irmãos, filhos de um determinado casal. Note que apesar da grande flexibilidade das árvores genéricas, que permitem inserir diversos filhos em relação a um mesmo nodo (ao contrário das árvores binárias), mesmo assim podem apresentar problemas para representar algumas relações mais complexas das famílias atuais e suas "recombinações exóticas". Outro exemplo de aplicação de uma árvore genérica seria a implementação de uma estrutura do tipo "sistema de arquivos" com uma hierarquia de diretórios (pais) e arquivos contidos nestes diretórios (diversos filhos associados a um diretório pai). Esta estrutura pode servir tanto para um sistema de arquivos quanto para guardar os "sites preferidos" (bookmarks) do Internet Explorer, que são armazenados em uma espécie de sistema de arquivos com pastas e sub-pastas.

As árvores genéricas podem ou não possuir uma ordenação em relação aos elementos nela inseridos, onde podemos ter uma árvore genérica sem nenhuma ordenação, como é o caso de uma "árvore genealógica" onde usualmente os nomes dos pais não influenciam nos nomes de seus filhos e nem os irmãos possuem alguma ordem de relação entre eles indicada pela ordenação de seus nomes. Entretanto, podemos imaginar uma árvore genérica usada para armazenar pastas (diretórios) e dados (arquivos) onde os nomes dos dados devam ser apresentados ao usuário de forma ordenada. Isto nos leva a seguinte pergunta: *Que tipo de rotinas devemos implementar para a inserção de nodos em uma árvore genérica?*

Vamos considerar as opções de inserção de um nodo em relação a hierarquia da árvore, resultado de sua relação com os demais nodos (pai, filho, irmão). Sendo assim, iremos trabalhar com 3 rotinas de inserção de dados em uma árvore genérica:

1. Insere um nodo filho em relação ao seu nodo pai: **insere filho**
2. Insere mais um nodo irmão em relação ao seu nodo irmão: **insere irmão**
3. Insere um nodo pai, acima de um nodo que ainda não possui pai: **insere pai**

Alguns cuidados devem ser tomados ao se implementar estas rotinas:

(i) Na inserção de mais um irmão, devemos decidir como iremos proceder em relação aos irmãos do nodo raiz: usualmente não se permite inserção de irmãos do nodo raiz da árvore, pois seria mais difícil de gerenciar esta lista de nodos irmãos, onde todos se encontram no topo da hierarquia da árvore. Ao contrário das demais listas de irmãos, não podemos subir um nível na árvore em relação a um nodo raiz, para depois descer e encontrar o primeiro filho deste nodo pai, sendo assim, se tivermos diversos irmãos-raiz, todos terão como pai um ponteiro para NIL. Em nossas rotinas vamos evitar esta situação.

(ii) Na inserção de um nodo pai em relação a outro já presente na árvore, devemos considerar que não poderemos alterar este nodo caso ele já possua um pai: qual seria a ação a executar caso um nodo que já possua um pai fosse ter um novo pai inserido? Como um nodo não pode ter 2 ou mais nodos pai, a solução que vamos adotar em nossas rotinas é a de permitir que esta inserção de um nodo pai só possa ser feita em relação ao nodo raiz, ou seja, em relação a um nodo que não possui pai (seu ponteiro pai aponta para NIL).

(iii) Em relação a ordenação, podemos fazer com que a inserção de nodos em um mesmo nível da hierarquia (inserção de irmãos, ou de mais um filho) seja realizada de forma ordenada. Considerando que a lista de irmãos é uma espécie de lista simplesmente encadeada (somente com o próximo irmão, apesar de todos estarem encadeados com o mesmo pai), basta que a inserção ocorra de modo ordenado: inserção ordenada em uma lista encadeada. Em nossas rotinas vamos adotar a solução mais simples, ou seja, inserção sem considerar a ordenação.

A remoção de nodos também deve ser bem planejada, pois remover um nodo folha é mais simples (não possui filhos), mas caso ele possua filho, como podemos fazer, uma vez que todos os seus filhos se tornariam órfãos? Uma solução pode ser implementar uma rotina **remove nodo**, onde esta rotina só permite a remoção de nodos que não possuem filhos (solução adotada), ou, remover toda a sub-arvore ligada a este nodo que está sendo removido.

Além das operações de inserção e remoção, outras funções muito importante no que se refere as árvores genéricas, são as operações de busca e exibição do conteúdo da árvore. Uma árvore genérica (ordenada ou não) poderá ser percorrida da mesma maneira que uma árvore binária:

- Percorrer de **modo préfixado**: Pré-Ordem
  Onde o VED = Visita/Esquerda/Direita passará a ser VIF = Visita, Irmãos e Filhos.
- Percorrer de **modo infixado** : Em Ordem
  Onde o EVD = Esquerda/Visita/Direita passará a ser IVF = Irmãos, Visita e Filhos.
- Percorrer de **modo pósfixado**: Pós-Ordem
  Onde o EDV = Esquerda/Direita/Visita passará a ser IFV = Irmãos, Filhos e Visita.

No caso da busca, não será mais possível otimizar a busca através de um processo de busca binária, pois a relação de ordem que permitia acelerar a busca não estará mais presente neste tipo de estrutura. Usaremos portanto o **modo de Busca Completa** (percorre todo os nodos), apenas optando se queremos implementar um procedimento que vai parar ao achar a primeira ocorrência do dado procurado (solução adotada), ou se ele realiza uma busca exaustiva em toda árvore, até encontrar todas as ocorrências existentes na árvore do dado buscado.

Vamos definir a seguir um conjunto de **rotinas genéricas** para manipulação de **Árvores Genéricas (AG = ArvGen)**, que simplificarão o seu uso e sua adaptação para outras aplicações. Foi definido nesta implementação a adoção dos critérios discutidos acima. Portanto o nodo da árvore genérica possuirá um ponteiro para o pai, para o seu primeiro filho (primeiro de uma lista) e para o seu próximo irmão (que pode formar uma outra lista). As rotinas ilustradas a seguir não irão tratar de: nodos ordenados ou duplicados; balanceamento de árvores genéricas; e não aceitarão a inserção de irmãos do nodo raiz bem como a inserção de um pai que não ocorra no próprio nodo raiz.

Definição dos Dados:

```
Const
        SEM_PAI = -1;           { Usado na Salva e Lê }

Type
         TArvGen_Dado       =  Integer;
         Ptr_ArvGen_Nodo    =  ^ArvGen_Nodo;
         ArvGen_Nodo        =  Record
                                  Dado: TArvGen_Dado;
                                  Pai, Filho, Irmao:  Ptr_ArvGen_Nodo;
                               End;
```

Definição das Rotinas:

```
{ Inicializa a árvore genérica – Prepara para inserir dados }
Procedure ArvGen_Inicializa (Var Raiz: Ptr_ArvGen_Nodo);
Begin
    Raiz := NIL;
End;

{ Insere um dado na árvore genérica – Insere como filho do nodo indicado }
Procedure ArvGen_Insere_Filho (Var Pai: Ptr_ArvGen_Nodo; Dado: TArvGen_Dado);
Var
  novo:Ptr_ArvGen_Nodo;
Begin
  new(novo);
  novo^.dado:=dado;
  novo^.filho:=NIL;
  novo^.irmao:=NIL;
  IF  Pai = NIL
  THEN Begin                    { Insere como Raiz da Arvore }
               Pai:=novo;
               novo^.pai:=NIL;
           End
  ELSE IF pai^.filho = NIL     { Insere o primeiro filho deste nodo }
         THEN Begin
                    Pai^.filho:=novo;
                    novo^.pai:=Pai;
                End
         ELSE  Begin                { Insere mais um filho deste nodo }
                    novo^.irmao:=Pai^.filho;
                    Pai^.filho:=novo;
                    novo^.pai:=Pai;
                End;
End;

{ Insere um dado na árvore genérica – Insere como irmão do nodo indicado }
{ Não permite a inserção de nodos irmãos em relação ao nodo raiz          }
Function ArvGen_Insere_Irmao (Var Filho: Ptr_ArvGen_Nodo; Dado: TArvGen_Dado):
Boolean;
Var
  novo:Ptr_ArvGen_Nodo;
Begin
    IF  Filho = NIL
    THEN Begin              { Insere como Raiz da Arvore }
                new(novo);
                novo^.dado:=dado;
                novo^.filho:=NIL;
                novo^.irmao:=NIL;
                Filho:=novo;
                novo^.pai:=NIL;
               ArvGen_Insere_Irmao:=True;
            End
```

```
   ELSE  IF  Filho^.pai = NIL                    { Erro: Tentativa de inserir um irmão no Raiz }
         THEN ArvGen_Insere_Irmao:=False
         ELSE  Begin                             { Insere mais um irmão deste nodo }
                  arvgen_insere_filho(Filho^.pai,Dado);
                  ArvGen_Insere_Irmao:=True;
               End;
End;


{ Insere um dado na árvore genérica – Insere como pai do nodo indicado     }
{ Só permite a inserção de nodos pais para quem não tem ainda pai => Raiz }
Function ArvGen_Insere_Pai (Var Filho: Ptr_ArvGen_Nodo; Dado: TArvGen_Dado):
Boolean;
 Var
    novo:Ptr_ArvGen_Nodo;
 Begin
     IF Filho^.pai = NIL            { Insere pai do raiz }
     THEN Begin
               new(novo);
               novo^.dado:=dado;
               novo^.filho:=filho;
               novo^.irmao:=NIL;
                novo^.pai:=NIL;
                Filho^.pai:=novo;
                ArvGen_Insere_Pai:=True;
            End
     ELSE  ArvGen_Insere_Pai:=False;   { Não pode inserir: já tem pai }
 End;


{ Exibe o conteúdo (dados) da árvore genérica – em modo pré-fixado }
Procedure ArvGen_Exibe_Prefixado (Raiz: Ptr_ArvGen_Nodo);
Begin
   IF  Raiz <> Nil
   THEN  Begin
               Write ('Nodo: ',Raiz^.dado:4,' - Pai: ');
               IF (Raiz^.pai <> NIL)
               THEN writeln (Raiz^.pai^.dado)
              ELSE writeln('(NIL - Este eh o nodo Raiz)');
               ArvGen_Exibe_Prefixado(Raiz^.irmao);
               ArvGen_Exibe_Prefixado(Raiz^.filho);
            End;
End;

{ Procura um dado na árvore e retorna um ponteiro para a primeira ocorrência deste dado }
{ ou retorna NIL se não achar }
Function  ArvGen_Pesquisa_Nodo (Raiz: Ptr_ArvGen_Nodo; Dado: TArvGen_Dado):
Ptr_ArvGen_Nodo;
Var
   Retorno:Ptr_ArvGen_Nodo;
```

```
Begin
      IF  Raiz <> NIL
      THEN  Begin
                  IF Raiz^.dado = dado
                 THEN  Retorno:=Raiz
                 ELSE  Begin
                             Retorno:=ArvGen_Pesquisa_Nodo(Raiz^.irmao,Dado);
                              IF  Retorno = NIL
                              THEN Retorno:=ArvGen_Pesquisa_Nodo(Raiz^.filho,Dado);
                            End;
            End
      ELSE  Retorno:=NIL;
      ArvGen_Pesquisa_Nodo:=Retorno;
End;

{ Salva em disco o conteúdo (dados) da árvore genérica }
Procedure ArvGen_Salva (Raiz: Ptr_ArvGen_Nodo);
Var
   ArqTxt:text;

  { Rotina interna da ArvGen_Salva: Salva percorrendo recursivamente a árvore }
  Procedure Salva_Recursiva(var Arq:Text; Raiz:Ptr_ArvGen_Nodo);
  Begin
      IF Raiz <> NIL
      THEN Begin
                  IF (Raiz^.pai <> NIL)
                  THEN writeln(Arq,Raiz^.pai^.dado)
                  ELSE writeln(Arq,SEM_PAI);
                  writeln(Arq,Raiz^.dado);
                  Salva_Recursiva(Arq,Raiz^.irmao);
                  Salva_Recursiva(Arq,Raiz^.filho);
              End;
  End;

Begin
   writeln('Salvando arquivo em disco...');
   assign(ArqTxt, 'arvgen.txt');                  { Nome do arquivo: ArvGen.txt }
   rewrite(ArqTxt);
   Salva_Recursiva(ArqTxt,Raiz);
   close(ArqTxt);
End;

{ -- Fim ArvGen Salva --}

{ Recupera do disco o conteúdo (dados) da árvore genérica }
Function  ArvGen_Le (Var Raiz: Ptr_ArvGen_Nodo): Boolean;
Var
   ArqTxt:Text;
   Pai,Filho:TArvGen_Dado;
   Ptr:Ptr_ArvGen_Nodo;
```

```
Begin
    IF Raiz <> NIL
    THEN ArvGen_Le:=false      { Arvore deve estar vazia }
    ELSE  Begin
              writeln('Lendo arquivo do disco...');
              arvgen_inicializa(Raiz);
              assign(ArqTxt, 'arvgen.txt');          { Nome do arquivo: ArvGen.txt }
              reset(ArqTxt);
              WHILE  Not  EoF (ArqTxt)
              DO  Begin
                        readln(ArqTxt,Pai);
                        readln(ArqTxt,Filho);
                        Ptr:=arvgen_pesquisa_nodo(Raiz,Pai);
                        arvgen_insere_filho(Ptr,Filho);
                        IF Raiz=NIL
                        THEN Raiz:=Ptr;
                    End;
              close(ArqTxt);
          End;
End;

{ Apaga toda a árvore genérica, liberando a memória ocupada }
Procedure ArvGen_Apaga (Var Raiz: Ptr_ArvGen_Nodo);
Begin
    IF  Raiz <> NIL
    THEN Begin
              ArvGen_Apaga(Raiz^.irmao);
              ArvGen_Apaga(Raiz^.filho);
              Dispose(Raiz);
         End;
    Raiz:=NIL;
End;

{ Remove o nodo cujo ponteiro foi passado como parâmetro }
Function ArvGen_Remove_Nodo (Var Nodo: Ptr_ArvGen_Nodo): Boolean;
Var
   Aux, OK:Ptr_ArvGen_Nodo;
Begin
    IF  Nodo = NIL    { Não tem nada para remover }
    THEN ArvGen_Remove_Nodo := False
   ELSE  Begin
              IF  Nodo^.filho <> NIL   { Se tem filhos... }
              THEN Begin
                        OK:=False;     { Não deixa remover o nodo }

                        {  ou então...
                           Apaga toda a sub-arvore associada a este nodo
                           ArvGen_Apaga(Nodo^.filho);
                           OK:=True;                                        }
                   End;
             ELSE   OK:=True;
```

```
    IF  OK
    THEN  Begin
                IF Nodo^.pai = nil      { Se não tem pai... é o raiz }
                THEN  Begin
                            dispose(Nodo);
                            Nodo:=nil;
                      End
                ELSE   Begin            { Se tem pai... pode ter irmão anterior }
                            IF Nodo^.pai^.filho = Nodo   { Primeiro Filho }
                            THEN Begin
                                        Nodo^.pai^.filho:=Nodo^.irmao;
                                        dispose(Nodo);
                                   End
                            ELSE   Begin       { Tem um irmao antes dele, acha! }
                                        aux:=Nodo^.pai^.filho; { 1o. filho }
                                        WHILE aux^.irmao <> Nodo
                                        DO aux:=aux^.irmao;
                                        aux^.irmao:=Nodo^.irmao;
                                        dispose(Nodo);
                                    End;
                      End;
                ArvGen_Remove_Nodo := True;
             End   { if ok }
  ELSE ArvGen_Remove_Nodo := False;
  End;  { if nodo = nil }
End;

{ Remove o nodo que contem o valor indicado como parâmetro,
  indicando se conseguiu remover }
Function ArvGen_Remove_Dado (Var Raiz: Ptr_ArvGen_Nodo; Dado: TArvGen_Dado):
Boolean;
Var
   nodo:Ptr_ArvGen_Nodo;
Begin
   nodo:=arvgen_pesquisa_nodo(Raiz,Dado);
   IF nodo = nil
   THEN  Begin
               writeln('ERRO: Nodo nao encontrado para remocao!');
                ArvGen_Remove_Dado:=False;
           End
   ELSE   Begin
                IF nodo=Raiz
                THEN Raiz:=nil;
                IF ArvGen_Remove_Nodo(nodo);
                THEN ArvGen_Remove_Dado:=True
               ELSE   ArvGen_Remove_Dado:=False;
           End;
End;
```

EXERCÍCIOS – *Árvores Genéricas*

1. Faça um programa para a manipulação de árvores genéricas usando as rotinas genéricas de manipulação de dados descritas acima. Leia um arquivo texto contendo uma seqüência de nomes e coloque os nomes em diferentes níveis da árvore de acordo com a inicial do nome. Exibir na tela a lista de nomes, agrupados de acordo com a letra inicial do nome.
   Exemplo:

   A
   Andréia
   Ana
   Alberto
   B
   Bianca
   C
   Carla
   Cristina
   ...

2. Faça um programa para a manipulação de árvores genéricas usando as rotinas genéricas de manipulação de dados descritas acima. Leia um arquivo texto com uma seqüência de linhas compostas de um identificador e um dado, conforme o exemplo abaixo, e gere uma árvore genérica onde o identificador de cada linha serve para indicar onde esta linha será inserida na hierarquia da árvore. A seguir, gere um arquivo texto, percorrendo a árvore do modo prefixado (ordem a ser seguida: nodo, irmão e por último filho).

   Exemplo do arquivo de dados: (os dígitos de cada sub-item variam apenas de 1 a 9).

   1- Nome
   1.1- Sexo
   1.2- Idade
   1.3- Nacionalidade
   1.3.1- Local_de_Nascimento
   1.3.2- Estado
   1.3.3- CEP
   1.4- Profissão
   1.5- Estado_Civil
   1.6- Carteira_de_Identidade
   1.7- CPF
   1.4.1- Salario
   1.4.2- Cargo