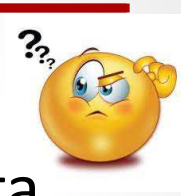
Estrutura de Dados

Árvores - Continuação

Profa. Dra. Lúcia Guimarães

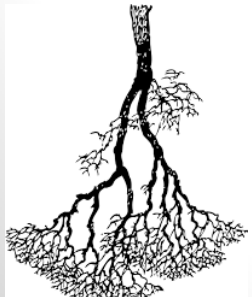


- **Arvores Binárias - Implementação**
  - Toda árvore Binária para ser implementada necessita de uma lei de formação
  - **Vamos pensar um pouco???**
    - Como poderíamos inserir um elemento numa árvore, considerando a lei de formação:
      - **Se Informação > Pai.Info**

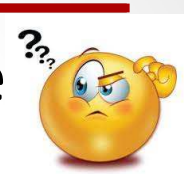
        ENTÃO Filho_Direita

        SENÃO Filho_Esquerda

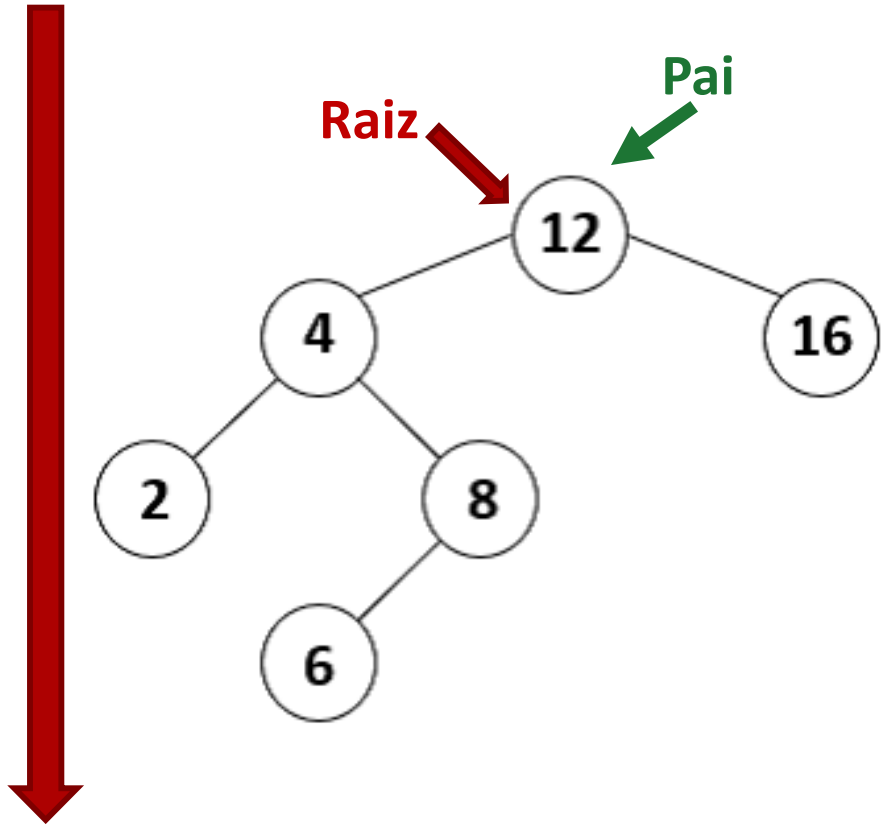
?????...



Arvores Binárias - Implementação

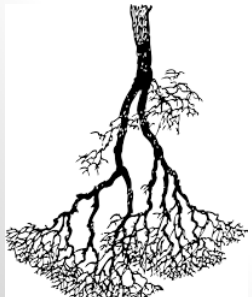


- Vamos pensar numa árvore para entender a lei de formação

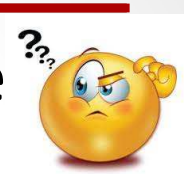


- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**

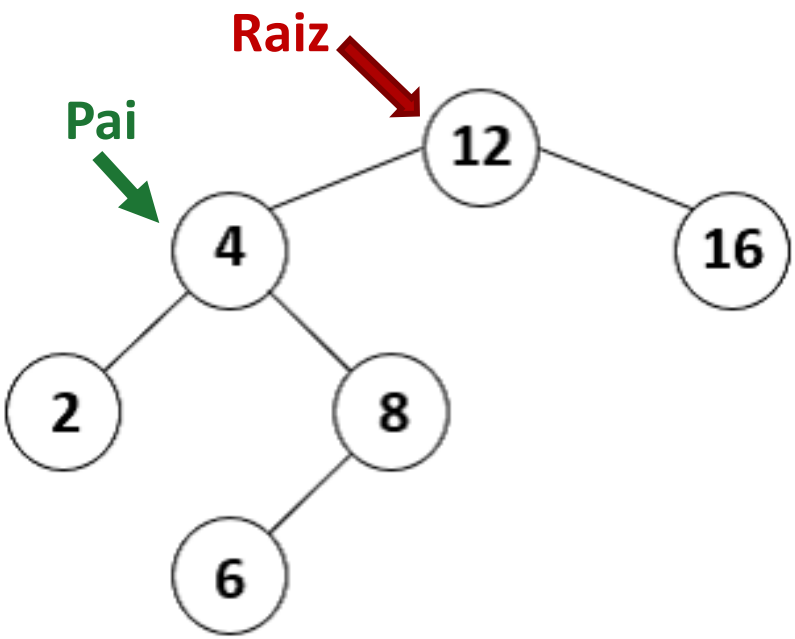
9 é maior que o pai = 12 ??



Arvores Binárias - Implementação

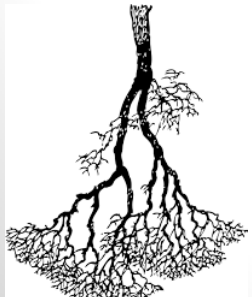


- Vamos pensar numa árvore para entender a lei de formação

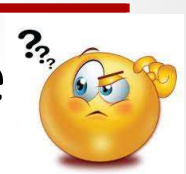


- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**

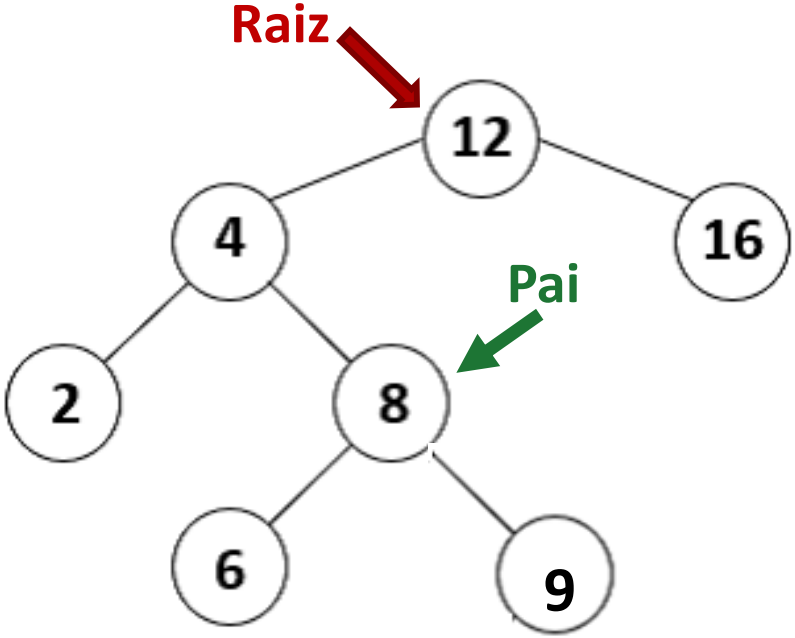
9 é maior que o pai = 4 ??



Arvores Binárias - Implementação

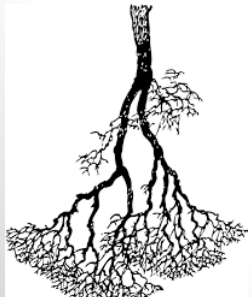


- Vamos pensar numa árvore para entender a lei de formação



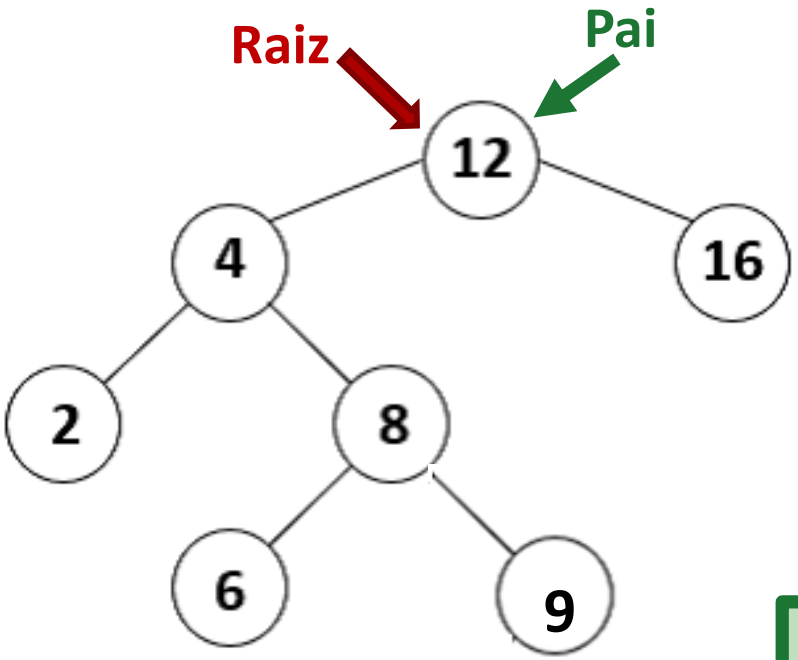
- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**

9 é maior que o pai = 8 ??



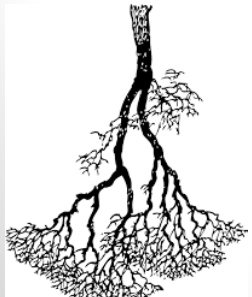
Arvores Binárias - Implementação

- Vamos pensar numa árvore para entender a lei de formação

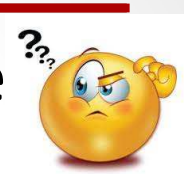


- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**
  - **15**

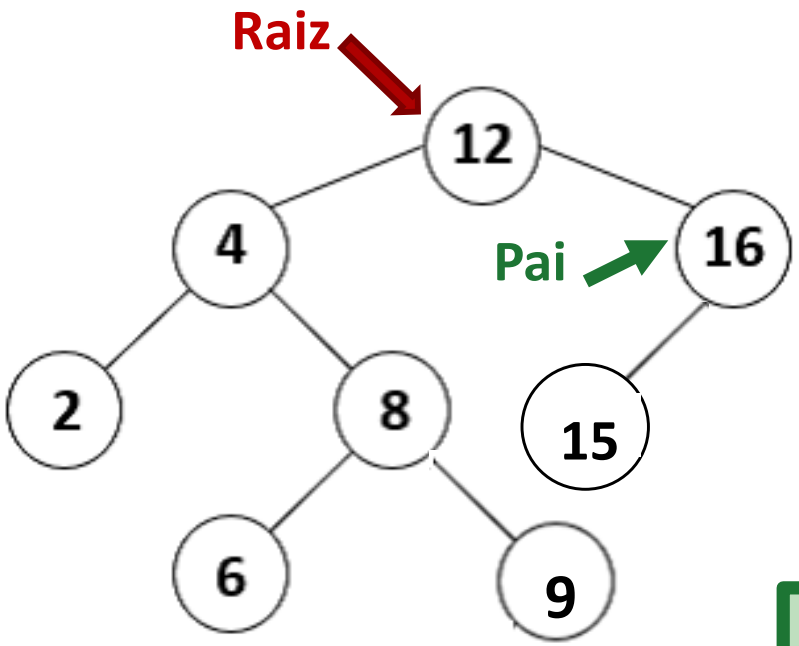
15 é maior que o pai = 12 ??



Arvores Binárias - Implementação

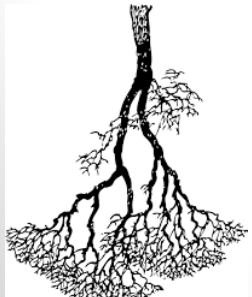


- Vamos pensar numa árvore para entender a lei de formação



- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**
  - **15**

15 é maior que o pai = 16 ??

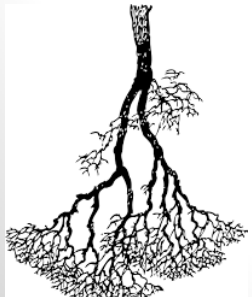


Arvores Binárias - Implementação

- **PseudoCódigo –**

ITERATIVO

```
leia (num)
Aloca espaço para novo
novo.info =num
novo Direita = ∅;
novo.Esquerda = ∅;
if(Raiz == ∅) então Raiz = novo
senão
    Pai = Raiz;
    flag =0;
    Enquanto(flag <>1) faça
        Se (Pai.info < num) então
                Se (Pai.Direita = ∅) então
                        Pai.Direita =novo;
                        flag= 1;
                            senão
                        Pai=Pai.Direita;
            senão
                Se (Pai.Esquerda = ∅ ) então
                        Pai.Esqueda = novo;
                         flag= 1;
                                    senão
                        Pai=Pai.Esquerda;
    Fim Enquanto
```

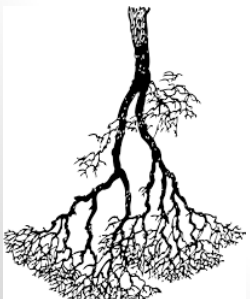


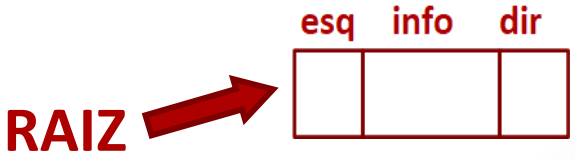
Arvores Binária - Implementação

- **Representação de um NÓ da árvore**:
  - Estrutura em C contendo
    - A **informação propriamente dita** (exemplo: um inteiro)
    - **Dois ponteiros** para as sub-árvores:
      - da esquerda e
      - da direita

- **Representação de uma ÁRVORE**:

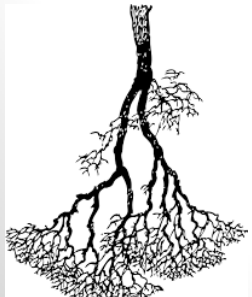
  Através de um **ponteiro para o nó raiz**





- **Estrutura em C contendo**
  - A **informação propriamente dita** (exemplo: um inteiro)
  - **Dois ponteiros** para as sub-árvores, da esquerda e da direita


```
typedef struct NoArvore
{
    int info;
    struct NoArvore *esq;
    struct NoArvore *dir;
}NoArv;
```

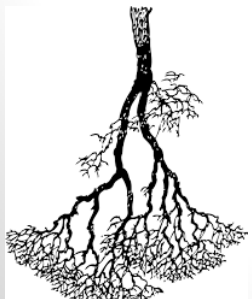


Arvores Binária - Implementação


```
typedef struct NoArvore
{
    int info;
    struct NoArvore *esq;
    struct NoArvore *dir;
}NoArv;
```

- **Criar uma estrutura para armazenar a raiz dessa árvore**

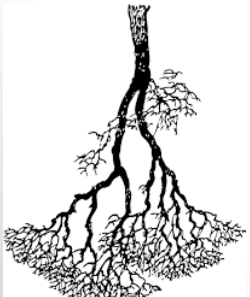
```
typedef struct Arvore
{
    NoArv *raiz;
}Arv;
```



Arvores Binária - Implementação

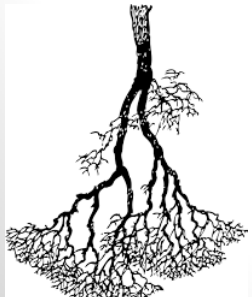
- **Funções Básicas para Manipulação de uma Árvore:**

  - **Inserir**
  - Verificar se o elemento existe na árvore (Busca)
  - Criar
  - Verificar se está vazia
  - Imprimir
  - Liberar
  - Remover



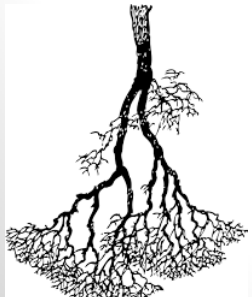
Funções Básicas para Manipulação de uma Árvore:

```
void insere(Arv *Arvore, int num)
{
    Arvore->raiz=aux_insere(Arvore->raiz,num);
}
```



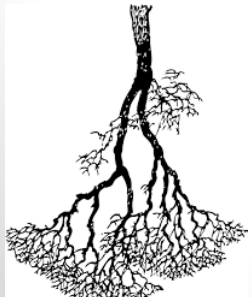
Funções Básicas para Manipulação de uma Árvore:

```
NoArv* aux insere(NoArv *no, int num)
{
    int flag;
    NoArv *Pai;
    NoArv *novo = (NoArv*)malloc(sizeof(NoArv));
    novo->info = num;
    novo->esq=NULL;
    novo->dir=NULL;
    if(no==NULL)
    {
        return novo;
    }
```



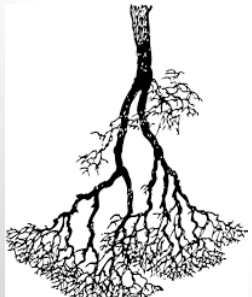
a Árvore:

```
    else
    {
        Pai = no;
        flag=0;
        while (flag==0)
        {
            if(Pai->info<num)
            {
                if(Pai->dir==NULL)
                {
                    Pai->dir = novo;
                    flag=1;
                }
                else
                {
                    Pai=Pai->dir;
                }
            }
            else
            {
                if(Pai->info>num)
                {
                    if(Pai->esq==NULL)
                    {
                        Pai->esq = novo;
                        flag=1;
                    }
                    else
                    {
                        Pai=Pai->esq;
                    }
                }
            }
        }
    }
    return no;
}
```



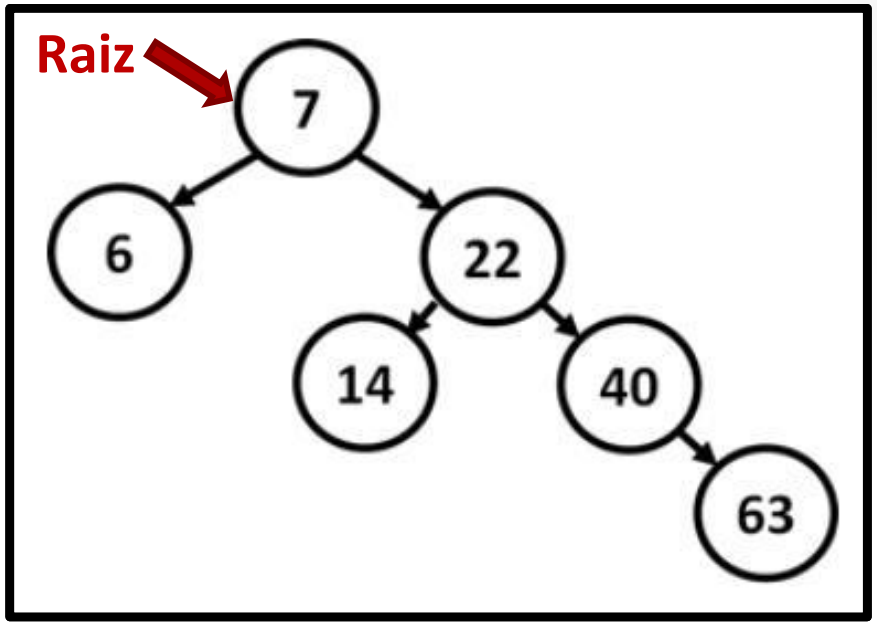
Arvores Binária - Implementação

- **Procedimentos de Percorrimento**
  - Tem por **finalidade percorrer a árvore como um todo.**
  - **Podem ser utilizados para imprimir a árvore**
  - **São três:**
    - **Pré-Order**
    - **In-Order**
    - **Pos-Order**

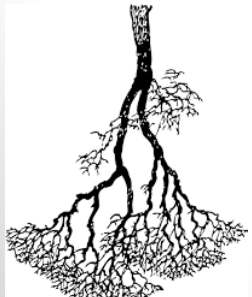


- **Procedimento Pré-Order - *Impressão***

```
PreOrder( Pai)
inicio
    imprime(Pai-info)
    Se (F_Direita≠∅) então
          PreOrder(F_Direita)
    Se (F_Esquerda≠∅) então
          PreOrder(F_Esquerda)
fim
```



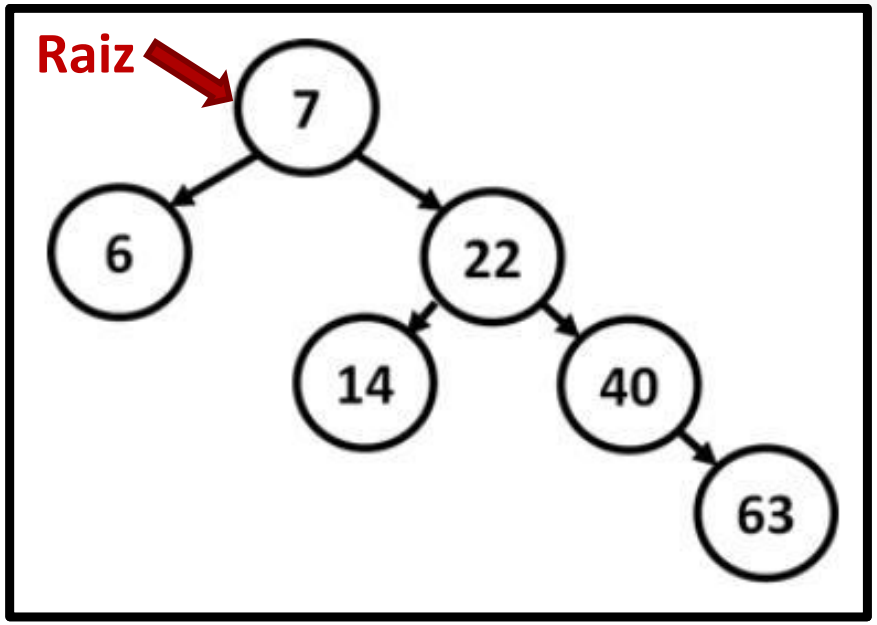
7 - 22 - 40 - 63 - 14 - 6

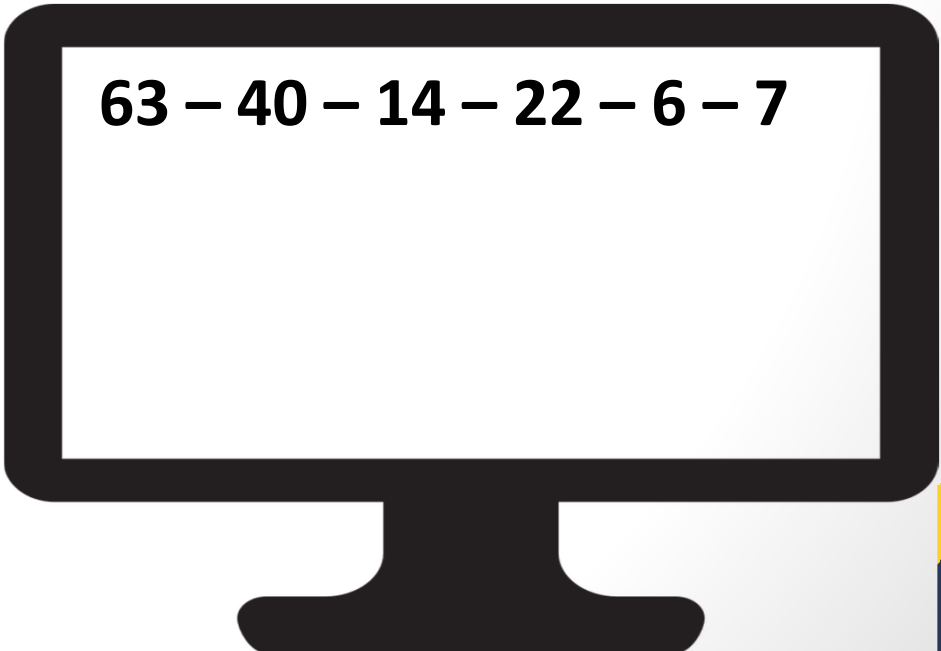


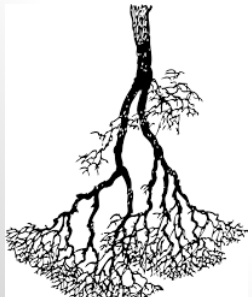
Arvores Binária - Implementação

- **Procedimento Pos-Order - *Impressão***

```
PosOrder( Pai)
inicio
   Se (F_Direita≠∅) então
         PosOrder(F_Direita)
   Se (F_Esquerda≠∅) então
         PosOrder(F_Esquerda)
    imprime(Pai-info)
fim
```



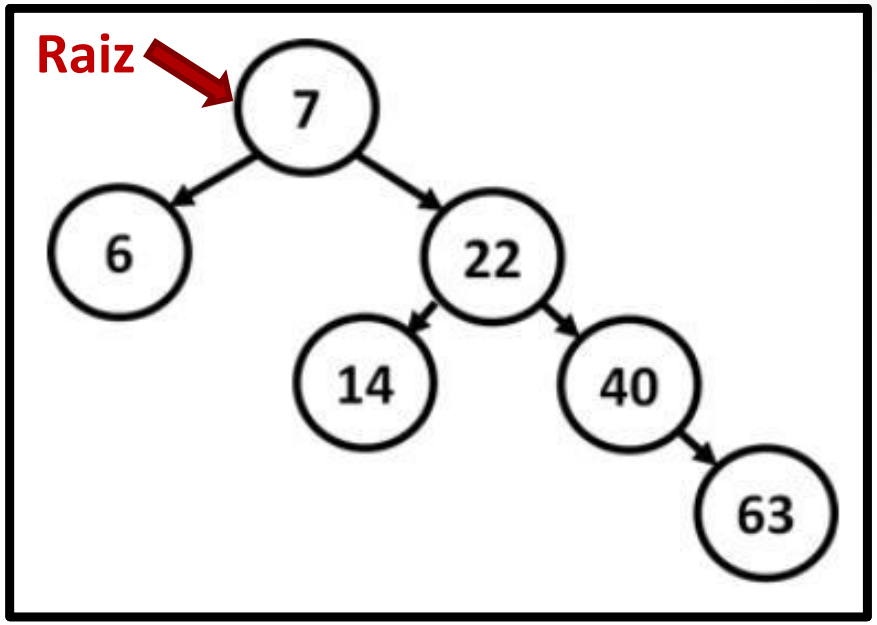


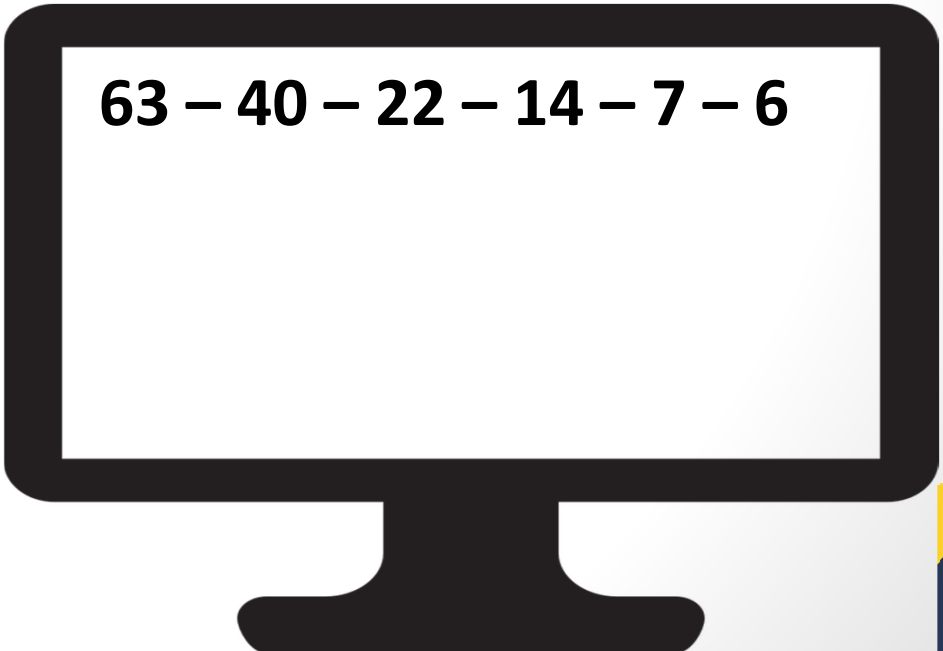


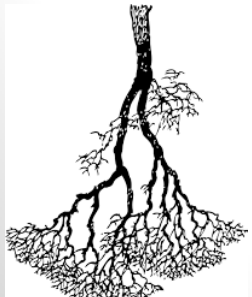
Arvores Binária - Implementação

- **Procedimento In-Order - *Impressão***

```
InOrder( Pai)
inicio
   Se (F_Direita≠∅) então
         InOrder(F_Direita)
    imprime(Pai-info)
   Se (F_Esquerda≠∅) então
         InOrder(F_Esquerda)
fim
```



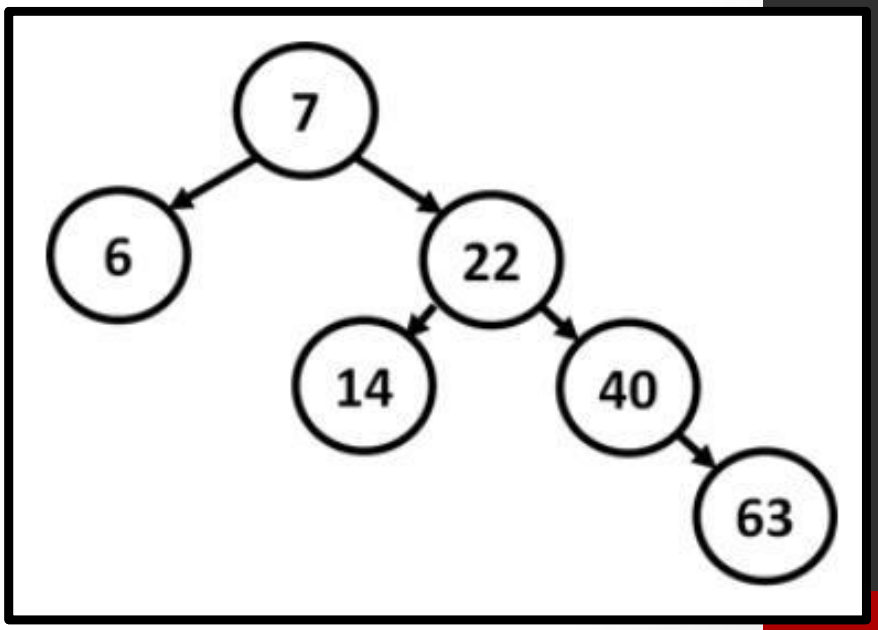


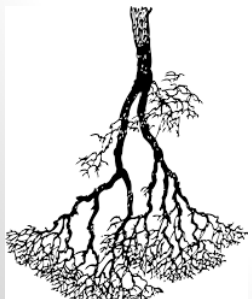


Arvores Binária - Implementação

- **Comentários para Elaboração de Procedimentos**

  - Basicamente todo procedimento de Árvore Binária pode ser desenvolvido pela adaptação de dois procedimentos básicos:
    - **Algoritmo de Busca ou Inserção**, onde a **lei de formação é conhecida** e usada para otimizar o procedimento
    - **Algoritmo de Percorrimento,** quando a árvore tem que ser **percorrida integralmente**





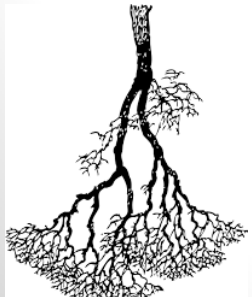
Arvores Binária - Implementação

- **Exercícios**

  - **Vamos começar a criar as funções que existiriam numa biblioteca árvore, considerando as seguintes estruturas:**

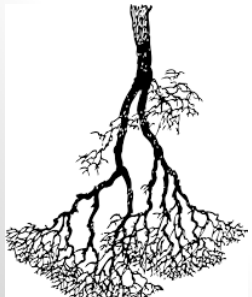
```
typedef struct NoArvore
{
      int info;
      struct NoArvore *esq;
      struct NoArvore *dir;
}NoArv;

typedef struct BaseArv
{
      NoArv *raiz;
}Arv;
```

- **Funções:**
  - **Inserir – já fizemos, modifique para não permitir elementos repetidos**
  - Cria_Arvore – aloca espaço para um ponteiro tipo Arv
  - Arv_Vazia – Retorna 1 se a arvore está vazia
  - Busca_Elemento – Retorna 1 se o elemento existir na árvore



Funções Básicas para Manipulação de uma Árvore:

- Criar uma árvore
- Verificar se a árvore está vazia



```c
typedef struct NoArvore
{
      int info;
      struct NoArvore *esq;
      struct NoArvore *dir;
}NoArv;
```

```c
typedef struct BaseArv
{
      NoArv *raiz;
}Arv;
```

ESTRUTURAS USADAS

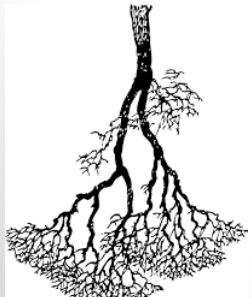
```c
Arv* Criar_Arvore()
{
    Arv *aux;
    aux=(Arv*)malloc(sizeof(Arv));
    aux->raiz = NULL;
    return aux;
}
```

```c
int ArvVazia(Arv *base)
{
    if(base->raiz==NULL)
    {
        return 1;
    }
    return 0;
}
```

Funções B

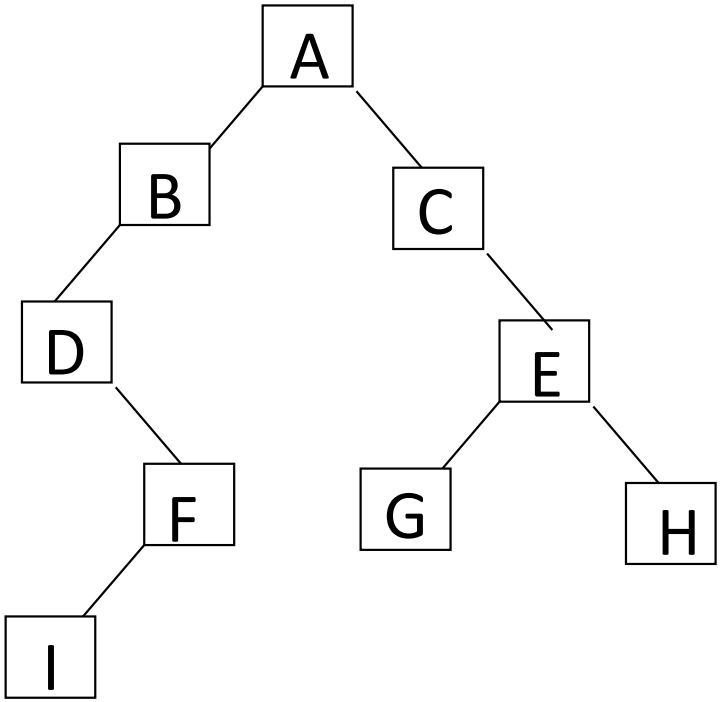
FALTA CONSTRUIR.....

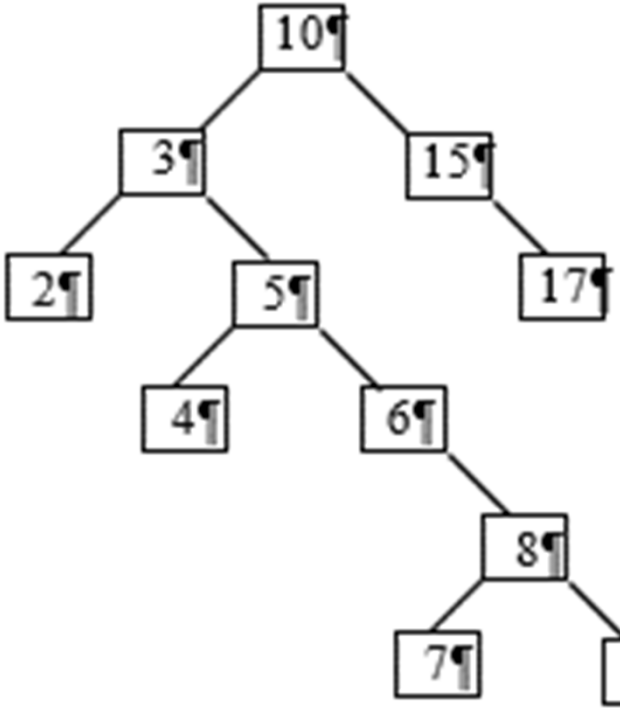
```c
int main()
{
    setlocale(LC_ALL,"portuguese");
    int i, num;
    Arv *RAIZ=NULL;
    RAIZ = Criar_Arvore();

    if(ArvVazia(RAIZ))
    {
        printf("\n\nÁRVORE VAZIA\n\n");
    }

    for(i=0; i<max; i++)
    {
        printf("\tDigite um número: ");
        scanf("%d",&num);
        insere(RAIZ,num);
    }
    if(!ArvVazia(RAIZ))
    {
        printf("\n\n\t\t ==> IMPRESSÃO\n\t");
        imprime_preOrder(RAIZ->raiz);
    }
    else
    {
        printf("\n\nÁRVORE VAZIA\n\n");
    }
    liberaArvore(RAIZ->raiz);
    free(RAIZ);
    RAIZ = NULL;
}
```

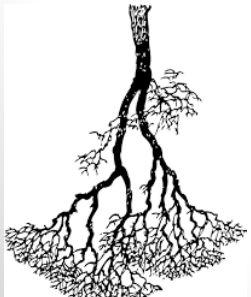


- **Exercícios**

1. Faça um teste de mesa para as árvores abaixo, usando os algoritmos pré, in e pós order







- **Exercícios**

2. Elabore um procedimento que

   a) Determine a soma de elementos de uma árvore

   b) Determine o número de ancestrais de uma determinada informação, se ela existir na árvore.

   c) Número de descendentes de um nó se ele existir na árvore

   d) Imprime o pai de um certo nó, se ele existir na árvore