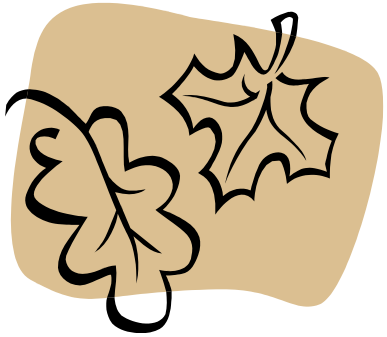


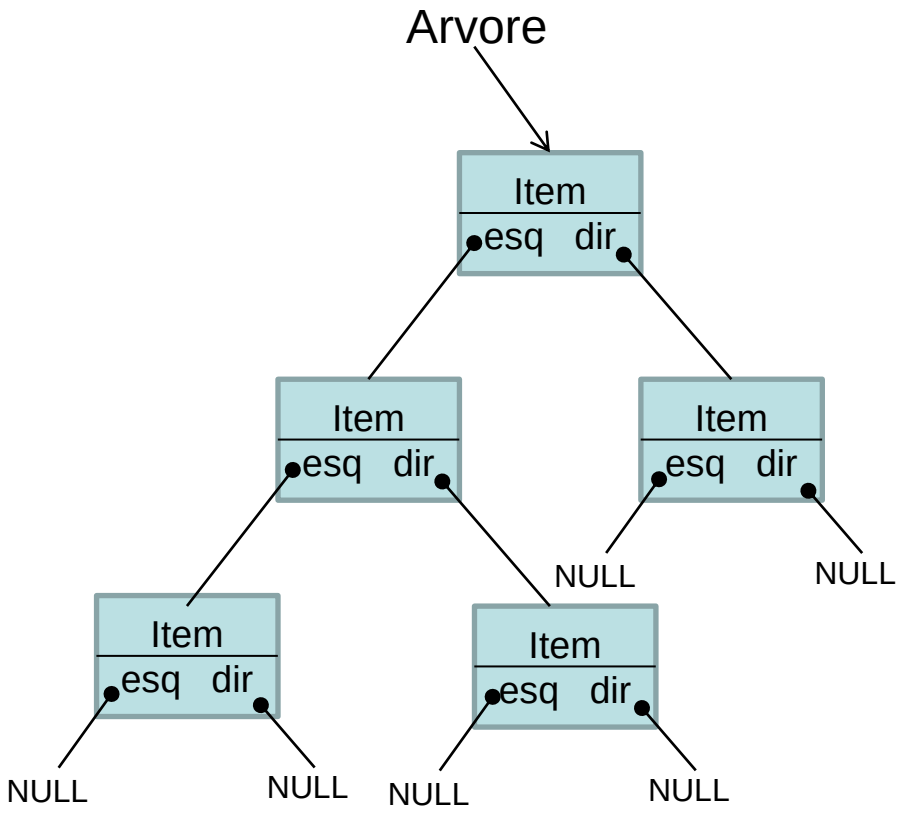
# ÁRVORE BINÁRIA DE BUSCA

IMPLEMENTAÇÃO

# Estrutura algorítmica

- A estrutura de dados para uma árvore binária é uma estrutura dinâmica, assim como as listas encadeadas, onde cada nó é representado por um registro contendo:
  - um campo *chave* do tipo inteiro, string, etc.
  - um ponteiro para a sub-árvore esquerda;
  - um ponteiro para a sub-árvore direita; e
  - outros campos de dados, de acordo com o problema de aplicação.

# Estrutura


Arvore
Item
esq dir
Item
esq dir
Item
esq dir
NULL
NULL
Item
esq dir
Item
esq dir
NULL
NULL
NULL
NULL

# Declarações

- TipoNo: registro (item: inteiro;
  TipoNo *esq;
  TipoNo *dir;);

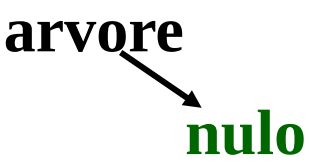
  arvore: ponteiro para TipoNo;

# Declaração em C++

```
Struct TipoNo{
        int item;
        TipoNo *esq;    //ponteiro para a subárvore esquerda
        TipoNo *dir;    //ponteiro para a subárvore direita
};

typedef TipoNo * TipoArvore;  //define um novo nome – tipoArvore – para o tipo ponteiro para nó.
```

# Inicialização




```
procedimento inicializa (var arvore: TipoArvore);
inicio
        arvore = nulo;
fim;
```

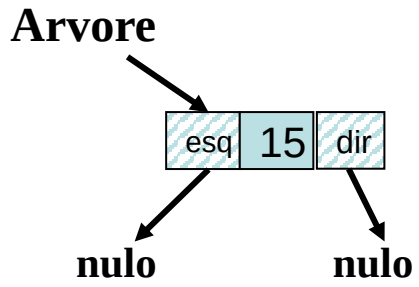
# Inicialização (C++)

```
//procedimento inicializaArvore


void inicializa(TipoArvore &arvore)
{
   arvore = NULL;
}
```

# Inserir um elemento na árvore

- Árvore vazia:
  - Se arvore está vazia
    - Cria um novo nó
    - Insere o elemento
    - Define ponteiros esquerdo e direito como nulo



# Inserir um elemento na árvore

```
Procedimento insereNaArvore(var arvore: TipoArvore; x: inteiro);
Inicio
        se (arvore == nulo) então
                inicializaPonteiro(arvore);
                arvore->item = x;
                arvore->esq = nulo;
                arvore->dir = nulo;
        fim-se;
        ...
        (continua... )
```

# Inserir um elemento na árvore

- Se a árvore não está vazia:
  - Compara com o elemento que está na raiz
    - Se for menor, insere na subárvore esquerda (recursivamente)
    - Se for maior, insere na subárvore direita (recursivamente)

# Inserir um elemento na árvore

```
(continuação ...)
        senão //árvore não está vazia
                se(x < arvore->item) então
                        insereNaArvore(arvore->esq, x);
                senão se (x > arvore->item) então
                        insereNaArvore(arvore->dir, x);
                senão
                        escreva (“Elemento já existe”);
                fim-se;
        fim-se;
fim;
```

# Inserir elementos (C++)

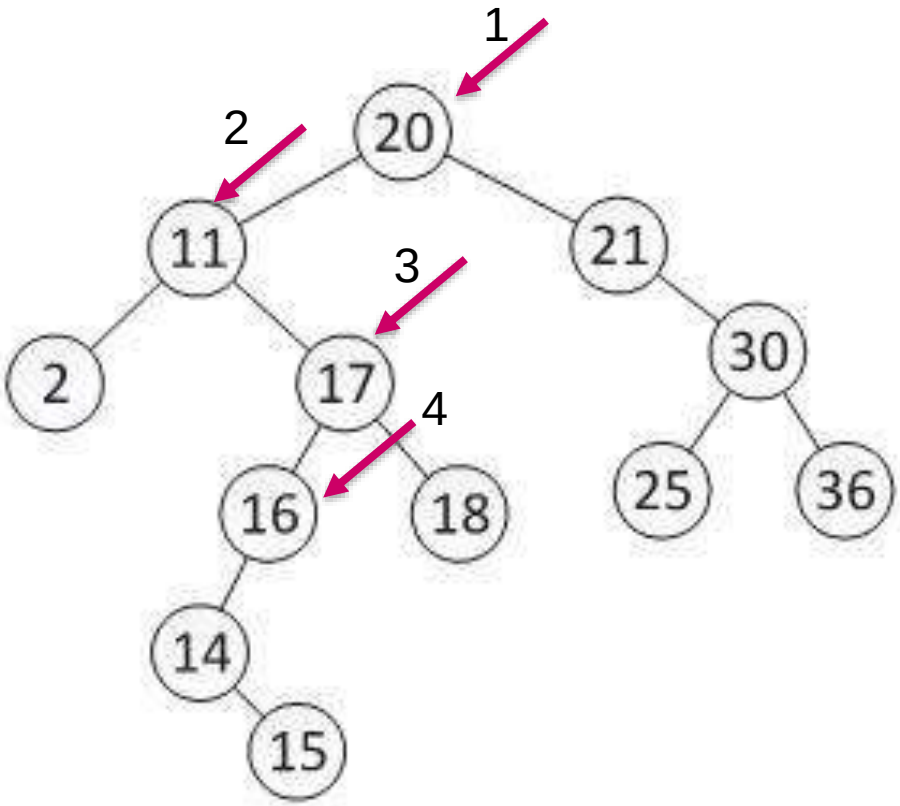
```
void insereNaArvore (TipoArvore &arvore, int x)
{
    if (arvore == NULL) //se a árvore está vazia
    {
        arvore = new TipoNo; //cria espaço na memória para o novo nó
        arvore->item = x;   //insere o item no campo item do nó
        arvore->esq = NULL;   //configura ponteiro esquerdo vazio
        arvore->dir = NULL;   //configura ponteiro esquerdo vazio
    }
    else (if x < arvore->item)
        insereNaArvore(arvore->esq, x);
    else (if x > arvore->item)
        insereNaArvore(arvore->dir, x);
    else
        cout << “Erro: elemento já existe na arvore.” << endl;
}
```

se a chave não está vazia, compara com a chave do nó para ver se o elemento a ser inserido é maior ou menor que esta.

Se a chave é menor que a chave armazenada no nó raiz, inicia-se novamente a busca (pelo local a ser inserido) na subárvore esquerda.

# Buscar um elemento

- Exemplo: buscar o elemento 16



# Buscar um elemento

```
Procedimento busca(arvore: TipoArvore; x: inteiro);
Inicio
        se (arvore == nulo) então
                escreva(“Elemento não existe”);
        senão se (arvore->item == x) então
                escreva(“Encontrado!”);
        senão se (x < arvore->item) então
                busca(arvore->esq, x);
        senão
                busca(arvore->dir, x);
        fim-se;
fim;
```

# Buscar um elemento

```
void busca (TipoArvore arvore, int x)
{
    if (arvore == NULL)  //arvore esta vazia ou o elemento não foi encontrado
            cout << “Elemento nao existe na arvore “<< endl;
    else (if x == arvore->item)
            cout << “Elemento encontrado! “ << endl;
    else (if x < arvore->item)
            busca(arvore->esq, x);
    else
            busca(arvore->dir, x);
}
```

# Imprimir os elementos da árvore

?

# Imprimir os elementos da árvore

## Percursos

Largura

Profundidade Pré-ordem

Em-ordem

Pós-ordem

# Percurso em Profundidade

- Pré-ordem (VED):

```
Procedimento preOrdem(arvore: TipoArvore);
inicio
        se arvore <> nulo então
                escreva(arvore->item);      //V
                preOrdem(arvore->esq);   //E
                preOrdem(arvore->dir);    //D
        fim-se;
Fim;
```

# Percurso em Profundidade (C++)

- Pré-ordem

```
void preOrdem(TipoArvore arvore)
{
        if (arvore != NULL)
        {
                cout << arvore->item << " ";
                preOrdem(arvore->esq);
                preOrdem(arvore->dir);
        }
}
```

# Percursos em Profundidade

- Em-ordem

```
Procedimento emOrdem(arvore: TipoArvore);
inicio
        se arvore <> nulo então
                emOrdem(arvore->esq);
                escreva(arvore->item);  //V
                emOrdem(arvore->dir);
        fim-se;
Fim;
```

# Percurso em Profundidade (C++)

- Em-ordem

```
void emOrdem(TipoArvore arvore)
{
        if (arvore != NULL)
        {
                emOrdem(arvore->esq);
                cout << arvore->item << " ";
                emOrdem(arvore->dir);
        }
}
```

# Percursos em Profundidade

- Pós-ordem

```
Procedimento posOrdem(arvore: TipoArvore);
inicio
        se arvore <> nulo então
                posOrdem(arvore->esq);
                posOrdem(arvore->dir);
                escreva(arvore->item);   //V
        fim-se;
Fim;
```

# Percurso em Profundidade (C++)

- Pós-ordem

```
void posOrdem(TipoArvore arvore)
{
        if (arvore != NULL)
        {
                posOrdem(arvore->esq);
                posOrdem(arvore->dir);
                cout << arvore->item << " ";
        }
}
```

# Percurso em Largura

- Utiliza uma estrutura FILA auxiliar para armazenar temporariamente os nós.
- À medida que vai passando pelos nós, vai inserindo na fila.
- A  impressão do percurso deve ser feita a partir da fila (desenfileirando os elementos e mostrando na tela).

```
Procedimento percursoEmLargura(arvore: TipoArvore)
Declare fila: TipoFila; //fila de ponteiros
        aux: ponteiro para TipoNó;
Inicio
        aux = arvore;
        se (aux <> nulo) então
                insereNaFila(aux);
                enquanto(fila não está vazia) faça
                        aux = removeDaFila(fila);
                        escreva( aux->item);
                        se (aux->esq <> nulo) então
                                insereNaFila(aux->esq);
                        fim-se;
                        se (aux->esq <> nulo) então
                                insereNaFila(aux->dir);
                        fim-se;
                fim-enquanto;
Fim;
```

C++

```
void percursoEmLargura(TipoArvore arvore) {
    queue<TipoArvore> fila;
    TipoNo *aux = arvore;
    if(aux!= NULL) {
        fila.push(aux);
        while(!fila.empty()) {
            aux = fila.front();
            fila.pop();
            cout << aux->item << " ";
            if(aux->esq != NULL)
                fila.push(aux->esq);
            if(aux->dir != NULL)
                fila.push(aux->dir);
        }
        cout << endl;
    }
}
```

# Exercícios

1. Fazer uma função para contar quantos elementos existem na árvore.

2. Fazer um procedimento para contar quantos números pares e quantos números ímpares existem na árvore.

3. Fazer um procedimento para separar uma árvore em duas, uma com os elementos pares e outra com os elementos ímpares.

# Links para os vídeos das aulas

- AED2
  - https://drive.google.com/file/d/1AeFcK5ZzYzvSwMfiUHa85OaNQrcdpXaJ/view?usp=sharing

- LAB2
  - https://drive.google.com/file/d/1Bf8YMroJoVSOWXtOCL17HLtVSwLDexAC/view?usp=sharing