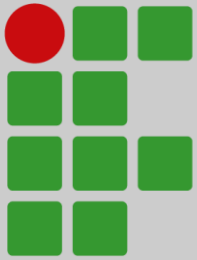


INSTITUTO FEDERAL

Campus Charqueadas

Árvores Binárias

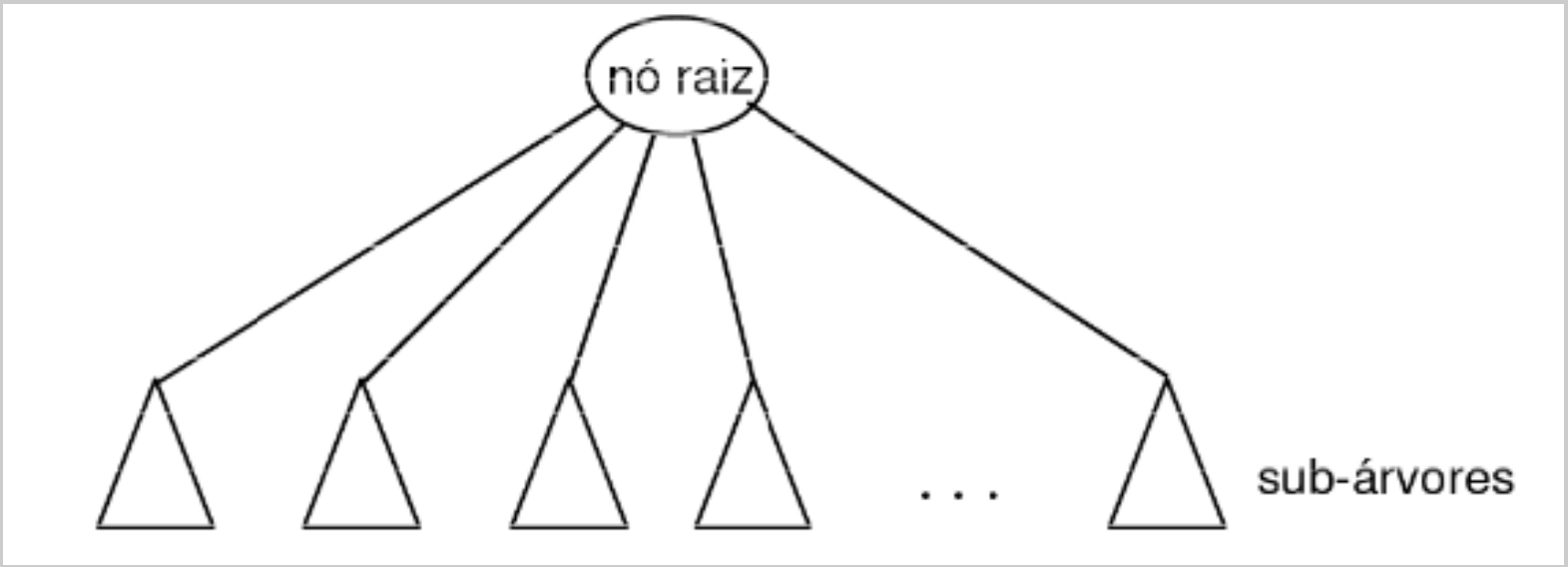
Prof. Silvana Teodoro

(silvanateo@gmail.com)

Árvores Binárias

Árvores Binárias

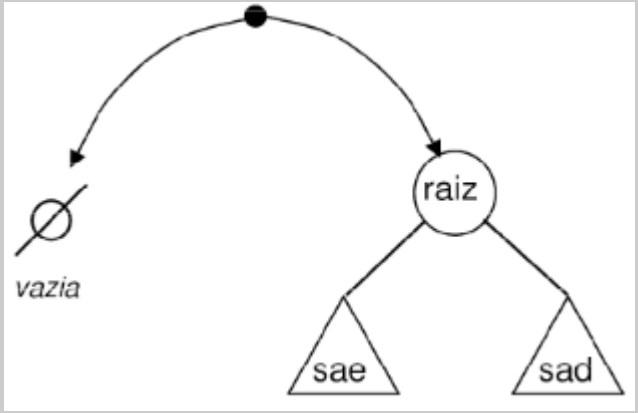
- Árvores são estruturas adequadas para representação de hierarquias.



Árvores Binárias

- **Uma árvore em que cada nó tem zero, um ou dois filhos**

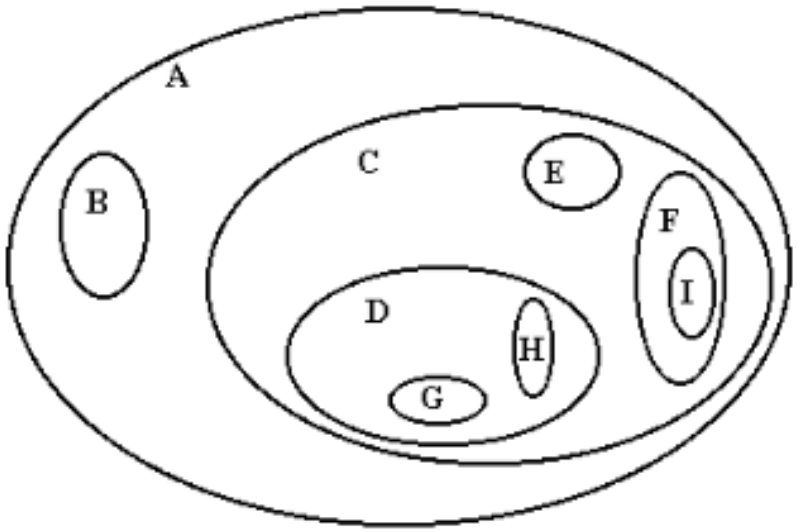
- **Uma árvore binária é:**
  - uma árvore vazia; ou
  - um nó raiz com duas sub-árvores:
    - a subárvore da direita (sad)
    - a subárvore da esquerda (sae)



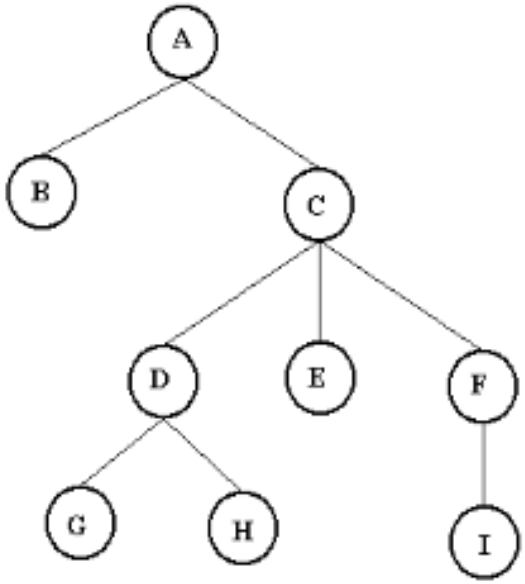
Árvores Binárias
Formas de Representação

- **Representação por parênteses aninhados**
  - (A (B) (C (D (G) (H)) (E) (F (I))))

Diagrama de Inclusão

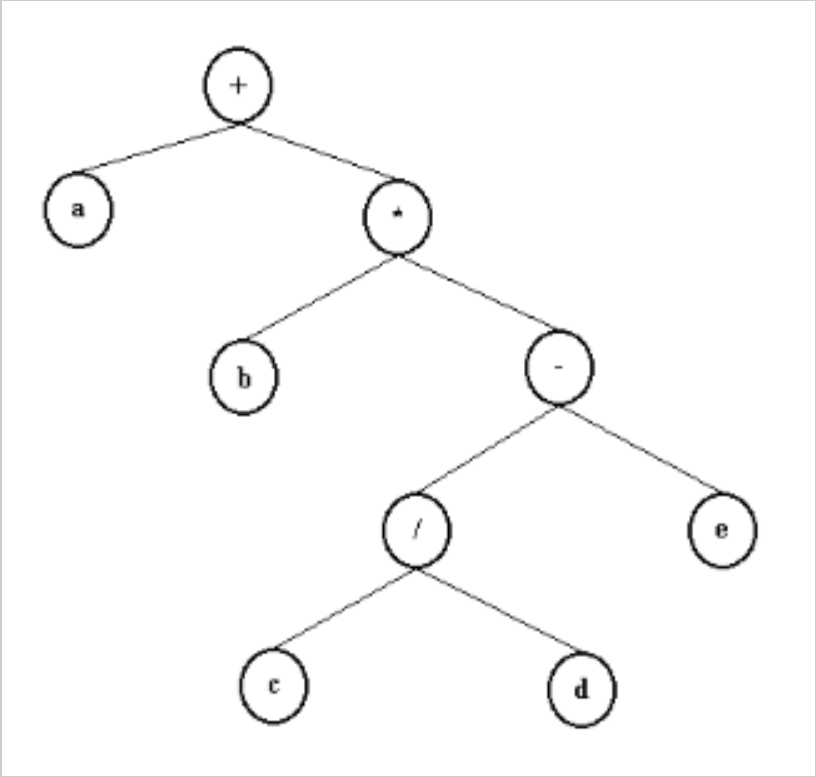


Representação Hierárquica



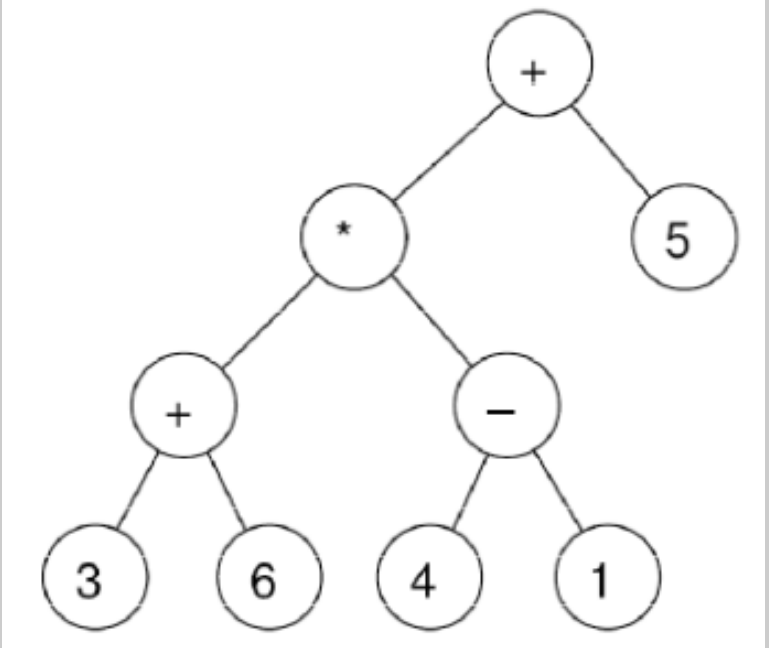
Árvores Binárias
Formas de Representação

- **Representação da expressão aritmética:**
  - (a + (b * (c / d - e)))



Árvores Binárias
Formas de Representação

- **Árvore binária representando expressões aritméticas binárias**
  - Nós folhas representam os operandos
  - Nós internos representam os operadores
  - (3+6)*(4-1)+5

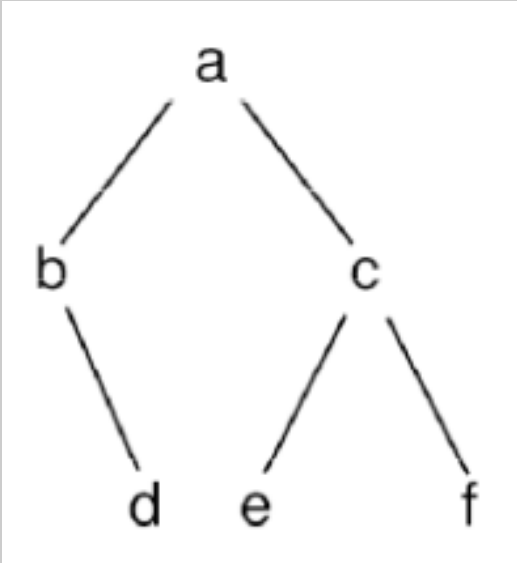


Árvores Binárias
Formas de Representação

- **Notação textual**
  - a árvore vazia é representada por <>
  - árvores não vazias por <raiz sae sad>

Exemplo:

- <a <b <> <d<><>> > <c <e<><>> <f<><> > > >



Árvores Binárias
Implementação em C

- **Representação: ponteiro para o nó raiz**
- **Representação de um nó na árvore:**
  - Estrutura em C contendo
  - A informação propriamente dita (exemplo: um caracere, ou inteiro)
  - Dois ponteiros para as sub-árvores, à esquerda e à direita

```
struct arv {
  char info;
  struct arv* esq;
  struct arv* dir;
};
```

Árvores Binárias
Implementação em C (arv.h)

```
typedef struct arv Arv;
//Cria uma árvore vazia
Arv* arv_criavazia (void);
//cria uma árvore com a informação do nó raiz c, e
//com subárvore esquerda e e subárvore direita d
Arv* arv_cria (char c, Arv* e, Arv* d);
//libera o espaço de memória ocupado pela árvore a
Arv* arv_libera (Arv* a);
//retorna true se a árvore estiver vazia e false
//caso contrário
int arv_vazia (Arv* a);
//indica a ocorrência (1) ou não (0) do caracter c
int arv_pertence (Arv* a, char c);
//imprime as informações dos nós da árvore
void arv_imprime (Arv* a);
```

Árvores Binárias
Implementação em C (arv.c)

- **Função arv_criavazia**
  - cria uma árvore vazia

```
Arv* arv_criavazia (void){
   return NULL;
}
```

Árvores Binárias
Implementação em C (arv.c)

- **Função arv_cria**
  - cria um nó raiz dadas a informação e as duas sub-árvores, a da esquerda e a da direita
  - retorna o endereço do nó raiz criado

```
Arv* arv_cria (char c, Arv* sae, Arv* sad){
  Arv* p=(Arv*)malloc(sizeof(Arv));
  p->info = c;
  p->esq = sae;
  p->dir = sad;
  return p;
}
```

Árvores Binárias
Implementação em C (arv.c)

- **arv_criavazia e arv_cria**
  - as duas funções para a criação de árvores
- **representam os dois casos da definição recursiva de árvore binária:**
  - uma árvore binária Arv* a;
  - é vazia a = arv_criavazia()
  - é composta por uma raiz e duas sub-árvores
  - é composta por uma raiz e duas sub-árvores
  - a = arv_cria(c,sae,sad);

Árvores Binárias
Implementação em C (arv.c)

- **Função arv_vazia**
  - indica se uma árvore é ou não vazia

```
int arv_vazia (Arv* a){
   return a==NULL;
}
```

Árvores Binárias
Implementação em C (arv.c)

- **Função arv_libera**
  - libera memória alocada pela estrutura da árvore
  - as sub-árvores devem ser liberadas antes de se liberar o nó raiz
  - retorna uma árvore vazia, representada por NULL

```
Arv* arv_libera (Arv* a){
  if (!arv_vazia(a)){
    arv_libera(a->esq); /* libera sae */
    arv_libera(a->dir); /* libera sad */
    free(a); /* libera raiz */
  }
  return NULL;
}
```

Árvores Binárias
Implementação em C (arv.c)

- **Função arv_pertence**
  - verifica a ocorrência de um caractere c em um dos nós
  - retorna um valor booleano (1 ou 0) indicando a ocorrência ou não do caractere na árvore

```
int arv_pertence (Arv* a, char c){
  if (arv_vazia(a))
      return 0; /* árvore vazia: não encontrou */
  else
      return a->info==c ||
      arv_pertence(a->esq,c) ||
      arv_pertence(a->dir,c);
}
```

Árvores Binárias
Implementação em C (arv.c)

- **Função arv_imprime**
  - percorre recursivamente a árvore, visitando todos os nós e imprimindo sua informação

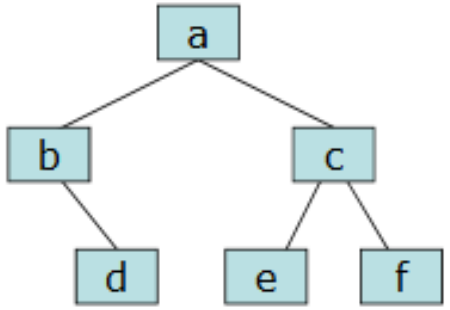
```
void arv_imprime (Arv* a){
  if (!arv_vazia(a)){
     printf("%c ", a->info); /* mostra raiz */
     arv_imprime(a->esq); /* mostra sae */
     arv_imprime(a->dir); /* mostra sad */
  }
}
```

Árvores Binárias
Implementação em C (arv.c)

- **Criar a árvore:**

<a <b <> <d <><>> > <c <e <><> > <f <><> > > >

```
/* sub-árvore 'd' */
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
/* sub-árvore 'b' */
Arv* a2= arv_cria('b',arv_criavazia(),a1);
/* sub-árvore 'e' */
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
/* sub-árvore 'f' */
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
/* sub-árvore 'c' */
Arv* a5= arv_cria('c',a3,a4);
/* árvore 'a' */
Arv* a = arv_cria('a',a2,a5 );
```

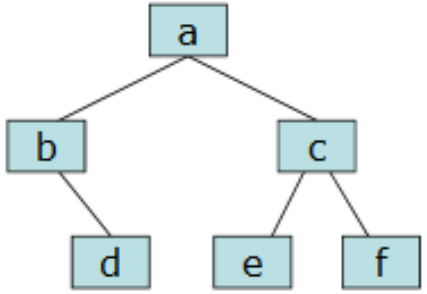


Árvores Binárias
Implementação em C (arv.c)

- **Criar a árvore:**

<a <b <> <d <><>> > <c <e <><> > <f <><> > > >

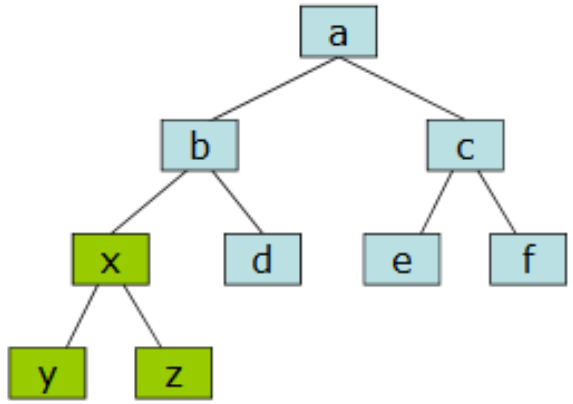
```
Arv* a = arv_cria('a',
           arv_cria('b',
               arv_criavazia(),
               arv_cria('d', arv_criavazia(), arv_criavazia())
           ),
           arv_cria('c',
              arv_cria('e', arv_criavazia(), arv_criavazia()),
              arv_cria('f', arv_criavazia(), arv_criavazia())
           )
);
```



Árvores Binárias
Implementação em C (arv.c)

- **Acrescenta nós x, y e z**

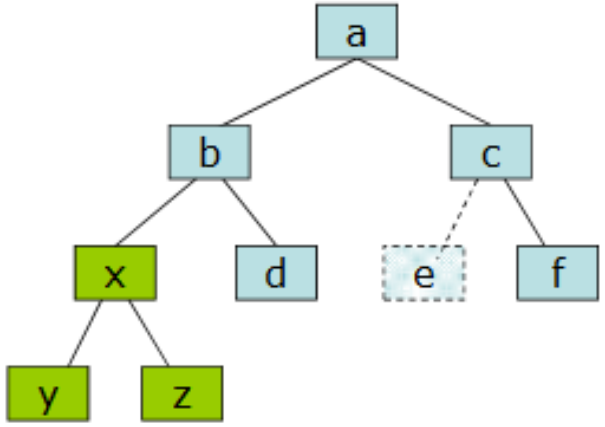
```
a->esq->esq =
  arv_cria('x',
      arv_cria('y',
            arv_criavazia(),
            arv_criavazia()),
      arv_cria('z',
            arv_criavazia(),
            arv_criavazia())
  );
```



Árvores Binárias
Implementação em C (arv.c)

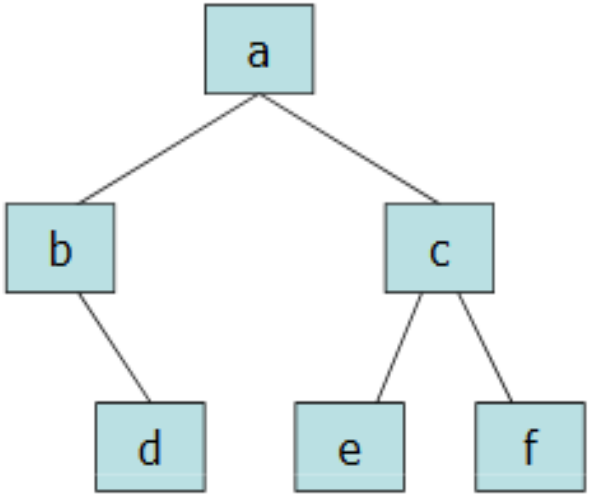
- **Libera nós**

```
a->dir->esq = arv_libera(a->dir->esq);
```



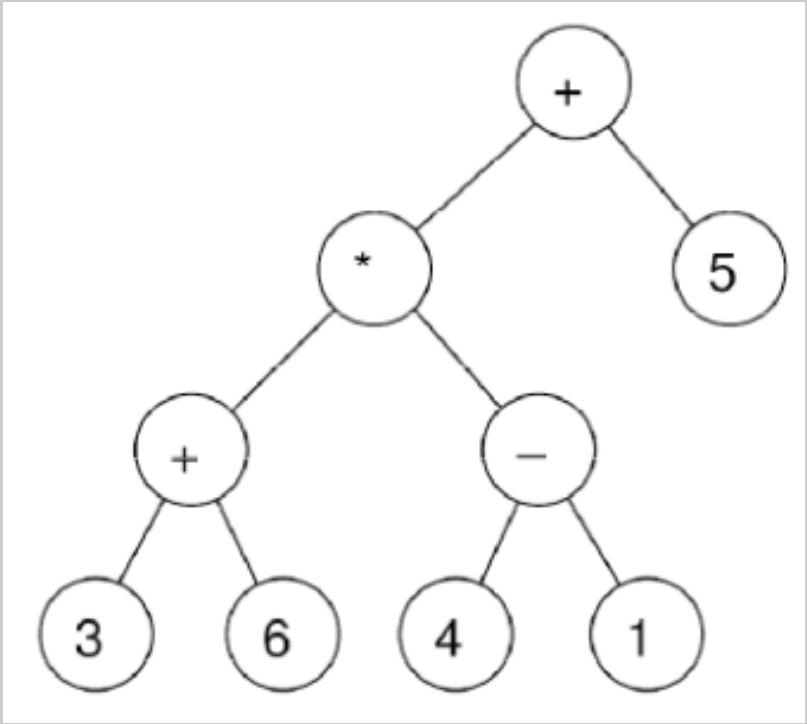
Árvores Binárias
Ordem de Percurso

- *Pré-ordem*:
  - trata *raiz*, percorre *sae*, percorre *sad*
  - exemplo: a b d c e f
- *Ordem simétrica (ou In-Ordem)*:
  - percorre *sae*, trata *raiz*, percorre *sad*
  - exemplo: b d a e c f
- *Pós-ordem*:
  - percorre *sae*, percorre *sad*, trata *raiz*
  - exemplo: d b e f c a



Árvores Binárias
Ordem de Percurso

- **Pré-ordem**
  - +*+36-415
- **In-ordem**
  - 3+6*4-1+5
- **Pós-ordem**
  - 36+41-*5+



Árvores Binárias
Ordem de Percurso

- **Pré-ordem**

```
void arv_preordem (Arv* a)
{
    if (!arv_vazia(a))
    {
        processa(a); // por exemplo imprime
        arv_preordem(a->esq);
        arv_preordem(a->dir);
    }
}
```

Árvores Binárias
Ordem de Percurso

- **In-ordem**

```
void arv_inordem (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_inordem (a->esq);
        processa (a); // por exemplo imprime
        arv_inordem (a->dir);
    }
}
```

Árvores Binárias
Ordem de Percurso

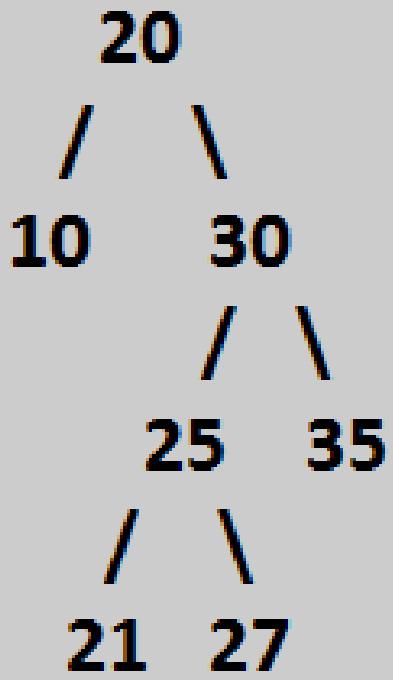
- **Pós-ordem**

```
void arv_posordem (Arv* a)
{
    if (!arv_vazia(a))
    {
        arv_posordem (a->esq);
        arv_posordem (a->dir);
        processa (a); // por exemplo imprime
    }
}
```

Exercícios

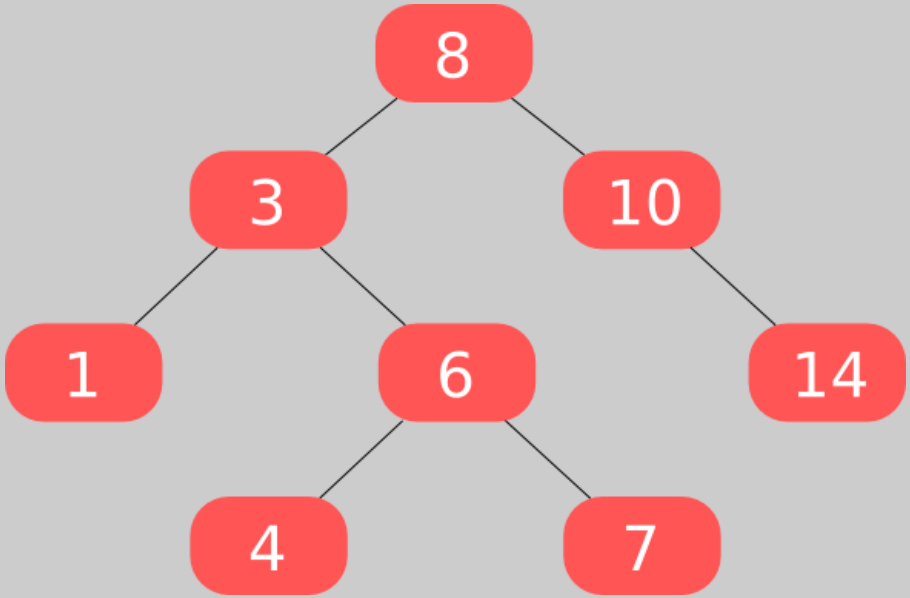
Exercícios

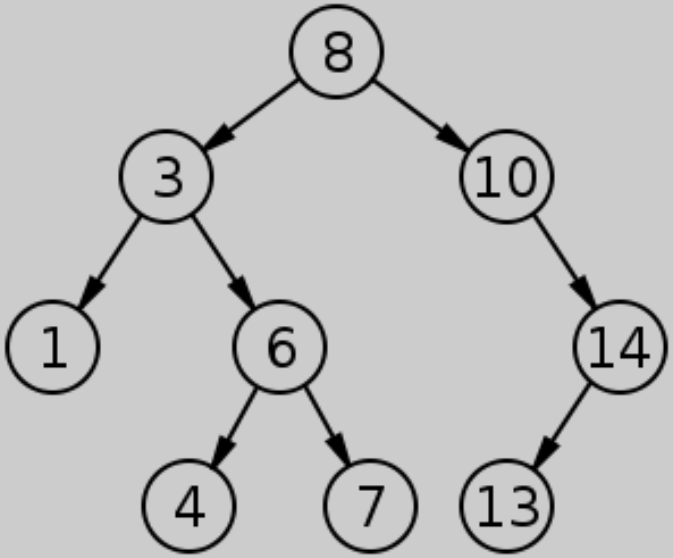
Represente a árvore binária abaixo usando a notação com parênteses aninhados.



Exercícios

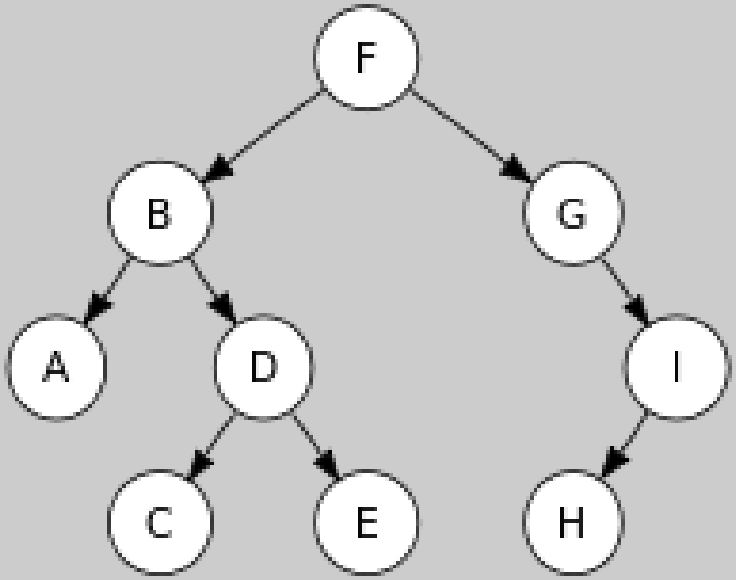
Escreva através de parênteses e notação textual as seguintes árvores:

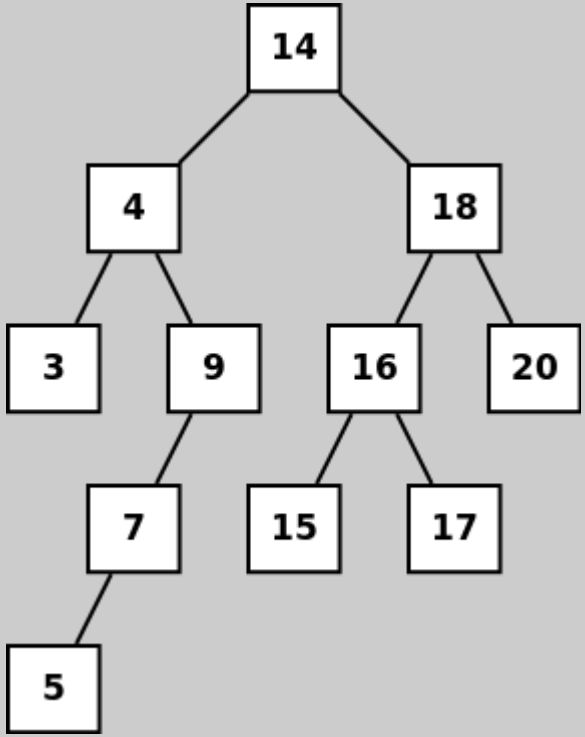




Exercícios

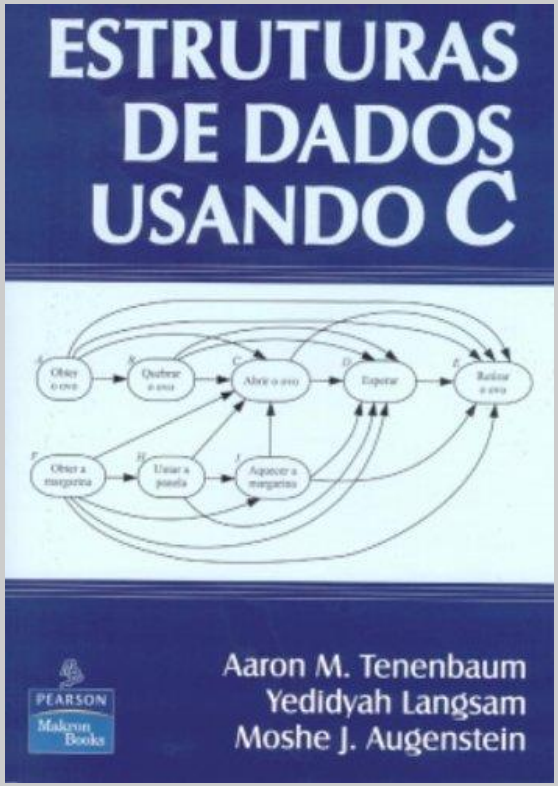
Escreva através de parênteses e notação textual as seguintes árvores:





Leitura Complementar

- Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein. **Estruturas de Dados Usando C.** Makron Books/Pearson Education, 1995.



Nas próximas aulas...

Implementação dos códigos vistos em aula.