

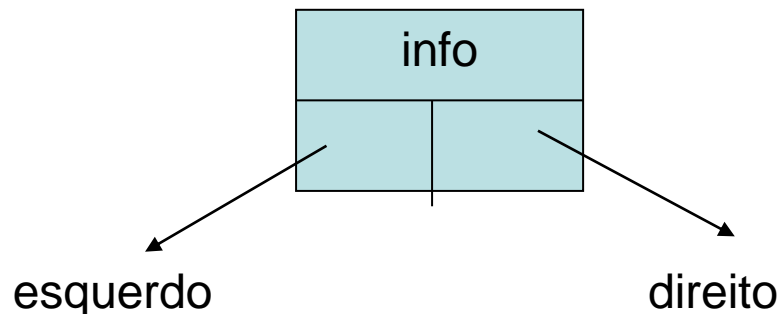
# Estruturas de Dados

## Árvores Binárias

Prof. Rosana

# Árvores Binárias

- É um tipo particular de árvore
- Cada item em uma árvore binária consiste de uma informação, e mais dois elos de ligação: um para o membro esquerdo e outro para o membro direito.



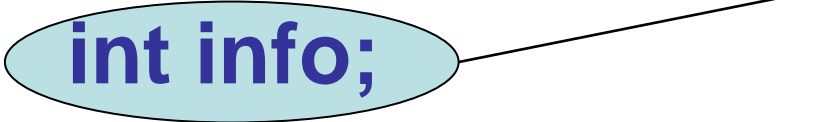
# Árvores Binárias de Busca

- As árvores binárias, **quando ordenadas**, conduzem a operações de busca muito rápidas.
- Nestes casos são chamadas de **árvores binárias de busca**.

# Árvores Binárias de Busca

- Definição básica:

```
struct arv{  
    int info;  
    struct arv *esq;  
    struct arv *dir;  
};  
typedef struct arv Arv;
```



ou a informação desejada

# ABB - Principais funções

//cria arvore vazia

Arv\* abb\_cria()

{

    return NULL;

}

//verifica se a árvore esta vazia

int abb\_vazia(Arv\* a)

{

    return (a == NULL);

}

(

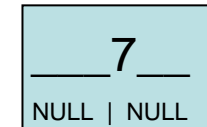
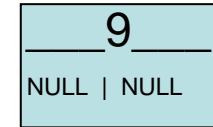
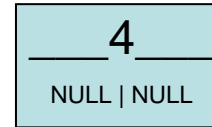
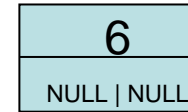
# ABB – Exemplo de main()

```
int main() {  
    int num;  
    Arv *a=abb_cria();  
    int v;  
    //insere elementos na abb até que um -1 seja digitado  
    do{  
        printf("Entre com o valor a ser inserido, ou -1 para finalizar:\n");  
        scanf("%d", &num);  
        if(num>=0)  
            a=abb_insere(a,num);  
    }while(num>=0);  
  
    abb_mostraEmOrdem2(a); //imprime em ordem  
    printf("\n\n");  
    system("pause");  
    return 0;  
}
```

# ABB - Principais funções (cont.)

// inserção interativa na ABB

```
Arv *abb_inserere(Arv* a,int c) {  
    Arv *p,*q,*r;  
    p=(Arv*)malloc(sizeof(Arv));  
    p->info = c;  
    p->esq = p->dir = NULL;  
  
    if (abb_vazia(a)) //1º elem. da árv.  
        a=p;  
    else {  
        q=a;  
        while(q!=NULL) {  
            r=q;  
            if(c < q->info)  
                q=q->esq;  
            else  
                q=q->dir;  
        }  
        if(c < r->info)  
            r->esq=p;  
        else  
            r->dir=p;  
    }  
    return a;  
}
```



# ABB - Principais funções (cont.)

//inserção recursiva na ABB

```
Arv* abb_inserere(Arv *a, int v){
```

```
    if (a==NULL){
```

```
        a=(Arv*) malloc(sizeof(Arv));
```

```
        a→info=v;
```

```
        a→esq=a→dir=NULL;
```

```
    }
```

```
    else if (v< a→info)
```

```
        a→esq=abb_inserere(a→esq,v);
```

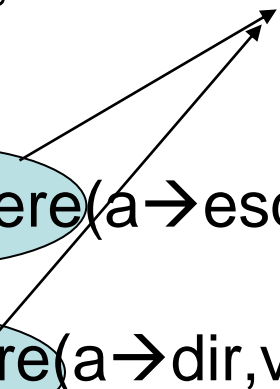
```
    else
```

```
        a→dir=abb_inserere(a→dir,v);
```

```
    return a;
```

```
}
```

***RECURSIVIDADE***





# ABB - Principais funções (cont.)

//busca um elemento na Abb

```
Arv* abb_busca(Arv *r, int v){  
    if (r==NULL) return NULL;  
    else if (r→info > v)  
        return abb_busca(r→esq,v);  
    else if (r→info < v)  
        return abb_busca(r→dir,v);  
    else  
        return r;  
}
```

//retorna o elemento ou NULL se não achou

# ABB - Principais funções (cont.)

//busca um elemento na Abb – outra versão

```
Arv* abb_busca(Arv *r, int v){  
    if (r==NULL) return NULL;  
    if (r→info > v)  
        return abb_busca(r→esq,v);  
    if (r→info < v)  
        return abb_busca(r→dir,v);  
    return r;  
}
```

# ABB – Funções de percurso

// percurso em ordem **esq – raiz - dir**

```
void abb_mostraEmOrdem(Arv *a){  
    if (a!=NULL){  
        abb_mostraEmOrdem(a→esq);  
        printf("%d\t", a→info);  
        abb_mostraEmOrdem(a→dir);  
    }  
}
```

//percurso em preordem **raiz – esq - dir**

```
void abb_mostraPreOrdem(Arv *a){  
    if (a!=NULL){  
        printf("%d\t", a→info);  
        abb_mostraPreOrdem(a→esq);  
        abb_mostraPreOrdem(a→dir);  
    }  
}
```

# ABB – Funções de percurso (cont.)

```
// percurso em PosOrdem  esq – dir - raiz
void abb_mostraPosOrdem(Arv *a){
    if (a!=NULL){
        abb_mostraPosOrdem(a→esq);
        abb_mostraPosOrdem(a→dir);
        printf("%d\t", a→info);
    }
}
```

# ABB - Principais funções (cont.)

```
Arv* abb_retira(Arv *r, int v) {  
    if (r==NULL) return NULL;  
    if(r->info > v) r->esq = abb_retira(r->esq, v);  
    else if (r->info < v) r->dir = abb_retira(r->dir, v);  
    else { //achou o elemento  
        if ( r->esq ==NULL && r->dir ==NULL) { // no sem filhos  
            free(r);  
            r=NULL; }  
        else if (r->esq ==NULL) { //filho a direita  
            Arv *t = r;  
            r=r->dir;  
            free(t); }  
        else if( r->dir ==NULL) { //filho a esquerda  
            Arv *t=r;  
            r=r->esq;  
            free(t); }  
        else { //tem 2 filhos  
            Arv *f=r->esq;  
            while (f->dir != NULL)f=f->dir;  
            r->info =f->info;  
            f->info=v;  
            r->esq=abb_retira(r->esq,v); }  
    }  
    return r;  
}
```

# ABB - Outras funções

//altura da arvore

```
int max2 (int a, int b)
{
    return (a>b)?a:b; //if ternário- retorna a se > ou b se <=
}

int abb_altura(Arv *a) //Número de níveis descontando a raiz
{
    if (abb_vazia(a))
        return -1;
    return 1+ max2(abb_altura(a->esq),abb_altura(a->dir));
}
```

# ABB – Exemplo de main()

```
int main() {  
    int num;  
    Arv *a=abb_cria();  
    int v;  
    //insere elementos na abb até que um -1 seja digitado  
    do{  
        printf("Entre com o valor a ser inserido, ou -1 para finalizar:\n");  
        scanf("%d", &num);  
        if(num>=0)  
            a=abb_insere(a,num);  
    }while(num>=0);  
  
    abb_mostraEmOrdem2(a); //imprime em ordem  
    printf("\n\n");  
    system("pause");  
    return 0;  
}
```

# Árvores binárias - considerações

- Uma árvore binária é uma forma especial de lista encadeada, porém com hierarquia.
- Pode-se inserir, acessar e remover itens em qualquer ordem. (Operações de recuperação não são destrutivas)
- A maioria das funções que usam árvores são **recursivas**, como a própria árvore.
- O processo de acesso a cada nó da árvore gera algoritmos diferentes, como os de busca em profundidade, busca em amplitude, etc...
- A **busca em profundidade** percorre todos os nós de um ramo (subárvore) até atingir os nós terminais, repetindo o processo em todos os ramos. (*preordenada*)
- Na **busca em amplitude** são visitadas a subárvore à esquerda, a raiz e a subárvore à direita, também repetindo o processo em todos os ramos.