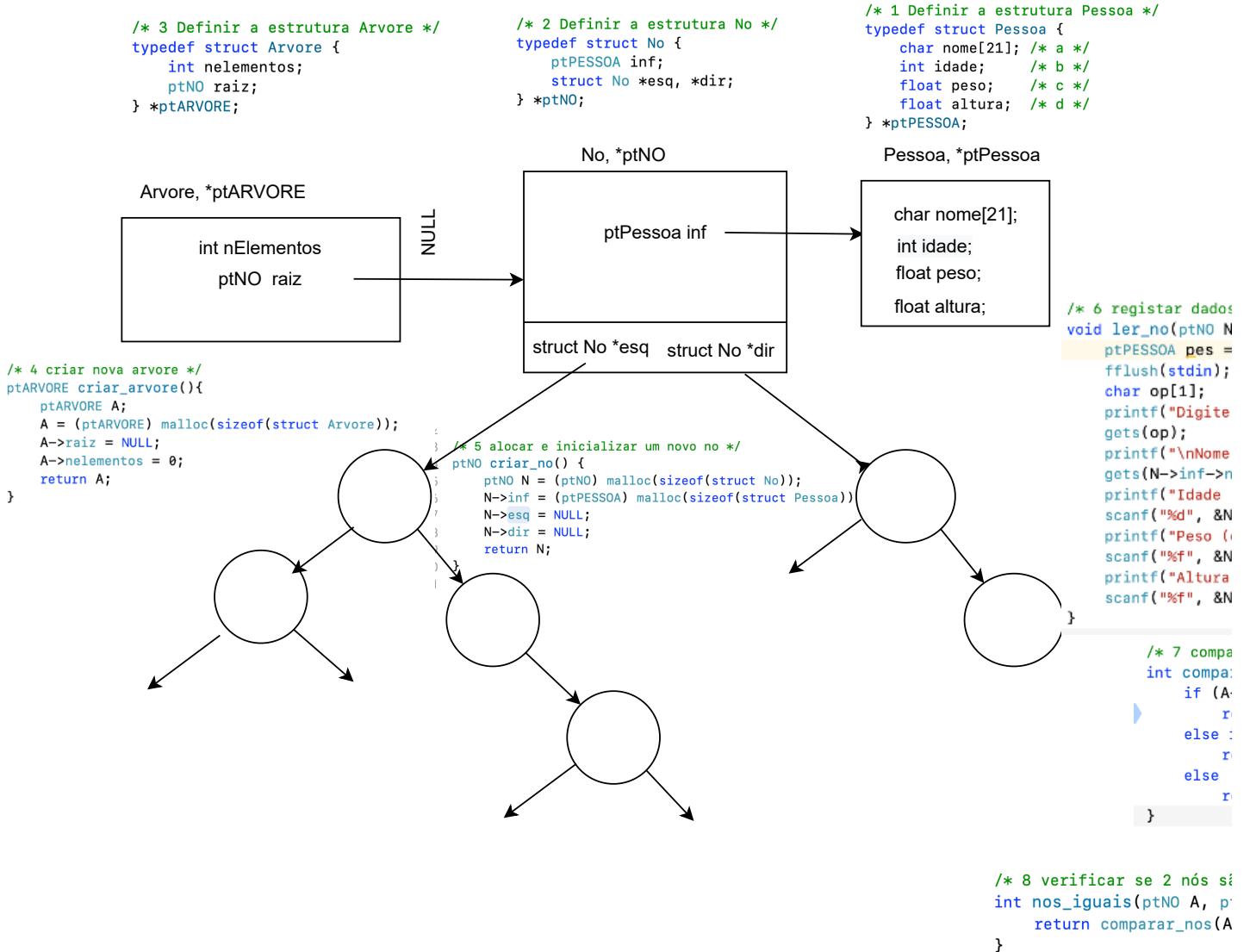


Panorama geral da Estrutura em Árvore e Algumas Funções Iniciais



S

```
s de um novo no */
}) {
= (ptPESSOA) N->inf; // pes aponta tb para a informação

: uma letra para iniciar a introduzir dados...");

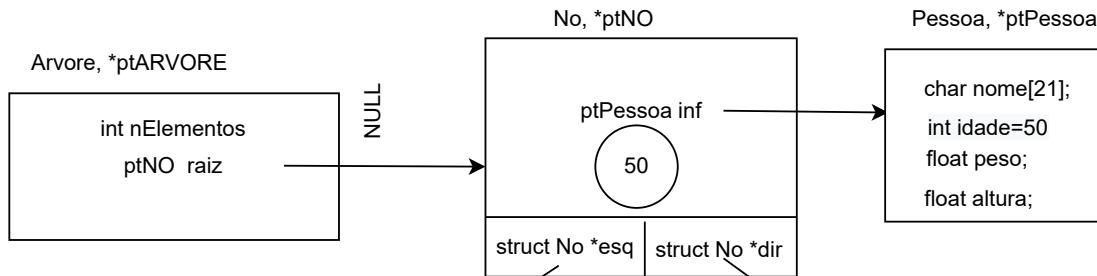
: ");
name);
(anos): ");
->inf->idade);
quilos): ");
->inf->peso);
(metros): ");
->inf->altura);

arar 2 nos (utilizando a idade) */
rar_nos(ptNO A, ptNO B){
->inf->idade < B->inf->idade)
return -1;
if (A->inf->idade > B->inf->idade)
return 1;

return 0; // s„o iguais

ão iguais (utilizando a idade) */
tNO B) {
\, B) == 0;
```

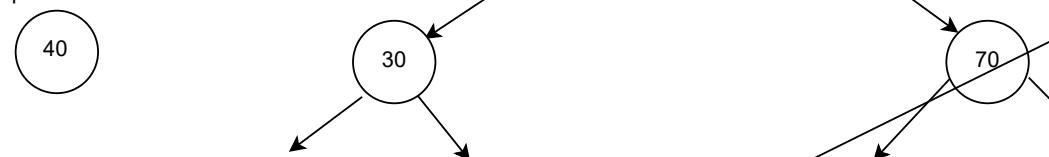
Inserir uma nova pessoa



```

    case 1: // inserir nó
    no_novo = criar_no();
    ler_no(no_novo);
    inserir_no(no_novo, arv);
    break; // -----
    
```

Inserir uma pessoa com 40 anos



```

/* 9 inserir novo nó na árvore (ordenação pelo campo idade) */
void inserir_no(ptNO no_novo, ptARVORE A) {
    if (!A) { printf("\nNão existe árvore... "); return; }
    if (!no_novo) { printf("\nNão há nó para adicionar... "); return; }
    if (A->raiz==NULL) // se a árvore está vazia
        A->raiz=no_novo; // o novo nó passa a ser a raiz da árvore
    else // se a árvore já contém elementos, vai inserir o novo nó no local apropriado
        insere(no_novo,A->raiz);
    A->nElementos++; // aumentar o numero de elementos
}
    
```

```

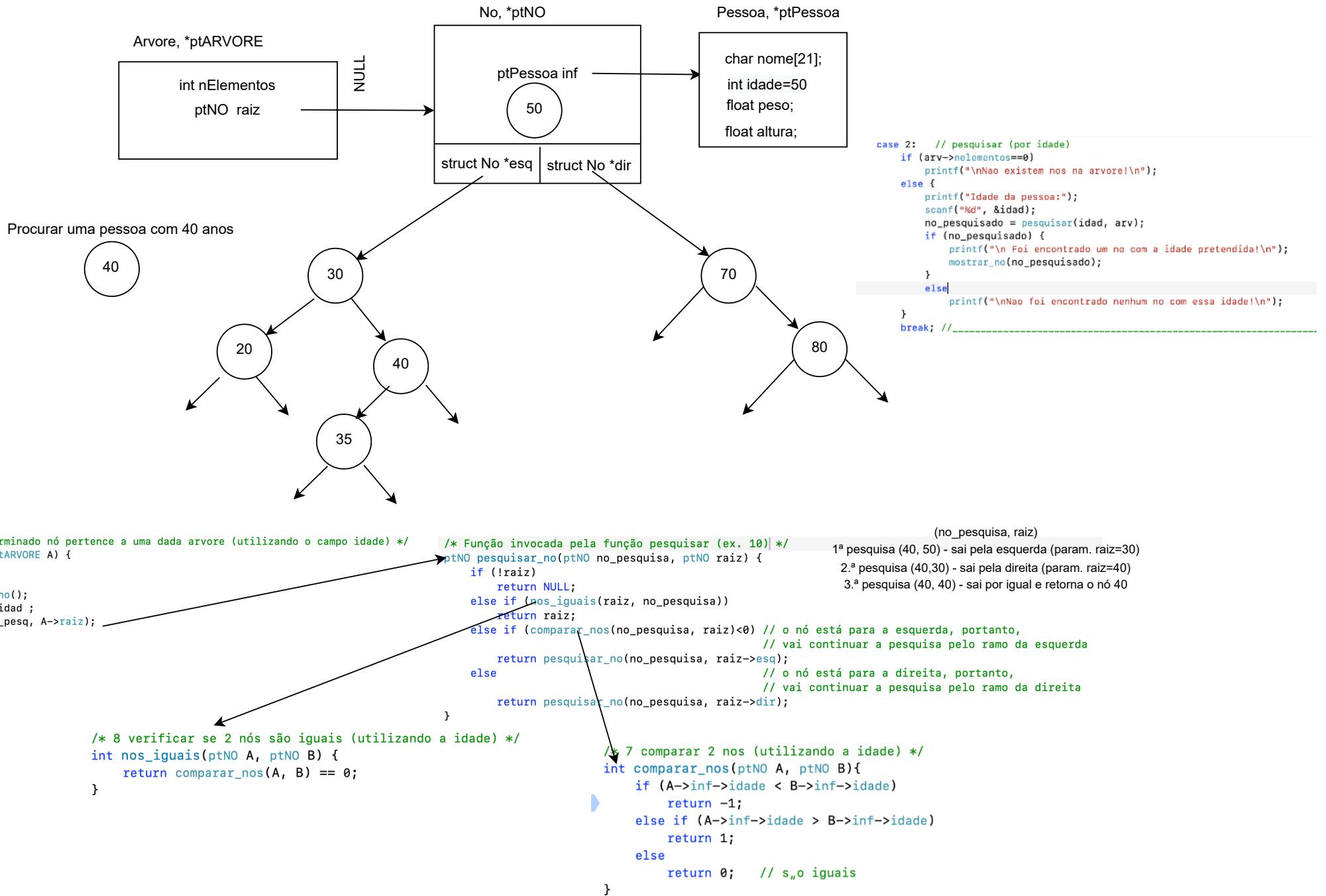
/* Função invocada pela função inserir_no (ex. 9) */
void insere(ptNO no_novo, ptNO raiz) {
    if (comparar_nos(no_novo, raiz) <= 0 ) // inserir à esquerda
        if ( !raiz->esq ) // se o ponteiro do raiz à esquerda estiver vazio,
            raiz->esq = no_novo; // é só inserir o novo nó
        else
            insere(no_novo,raiz->esq); // vai percorrer recursivamente o ramo esquerdo
                                            // até encontrar o final do ramo e depois colocar lá o novo nó
    else // inserir à direita
        if ( !(raiz->dir) )
            raiz->dir = no_novo;
        else
            insere(no_novo,raiz->dir);
}
    
```

```

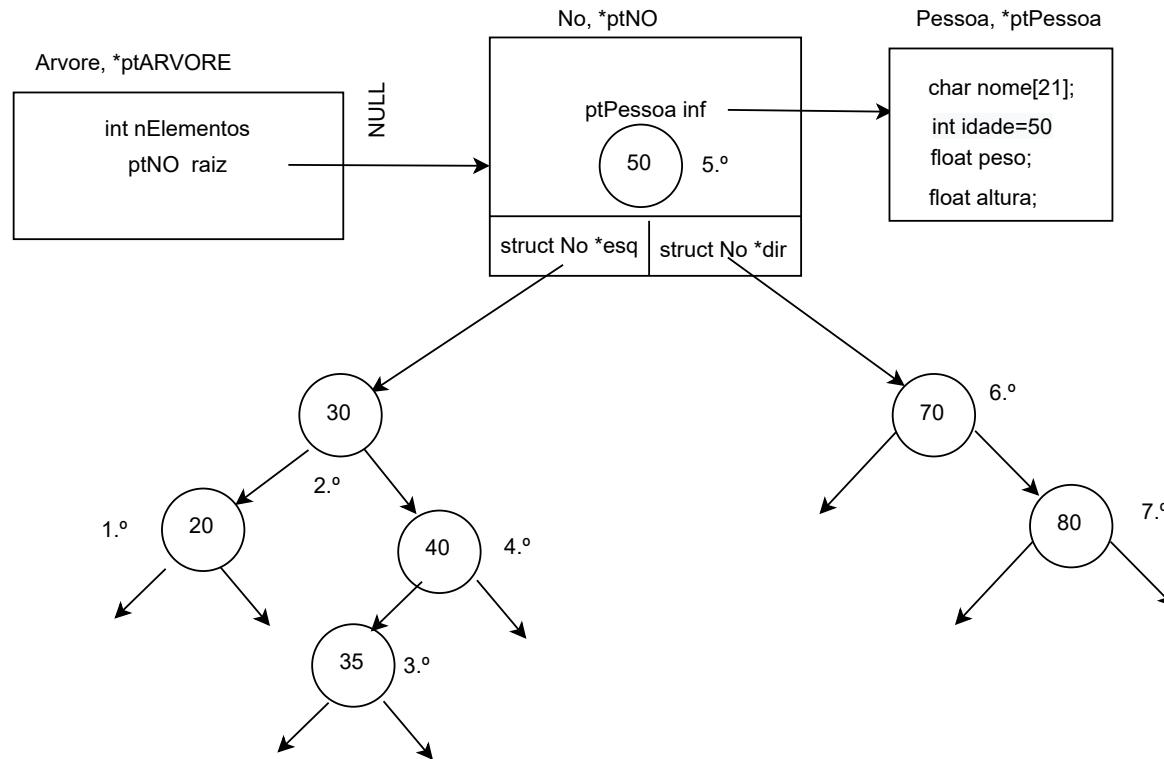
/* 7 comparar 2 nos (utilizando a idade) */
int comparar_nos(ptNO A, ptNO B){
    if (A->inf->idade < B->inf->idade)
        return -1;
    else if (A->inf->idade > B->inf->idade)
        return 1;
    else
        return 0; // são iguais
}
    
```

```
/* 10 verificar se
ptNO pesquisar(int
if (!A)
    return NUL
ptNO no_pesq =
no_pesq->inf->
return pesquis
}
```

Pesquisar um nó na árvore (procurar uma pessoa com uma dada idade)



Mostrar todos os Nós da árvore usando a técnica: (3) inorder



```

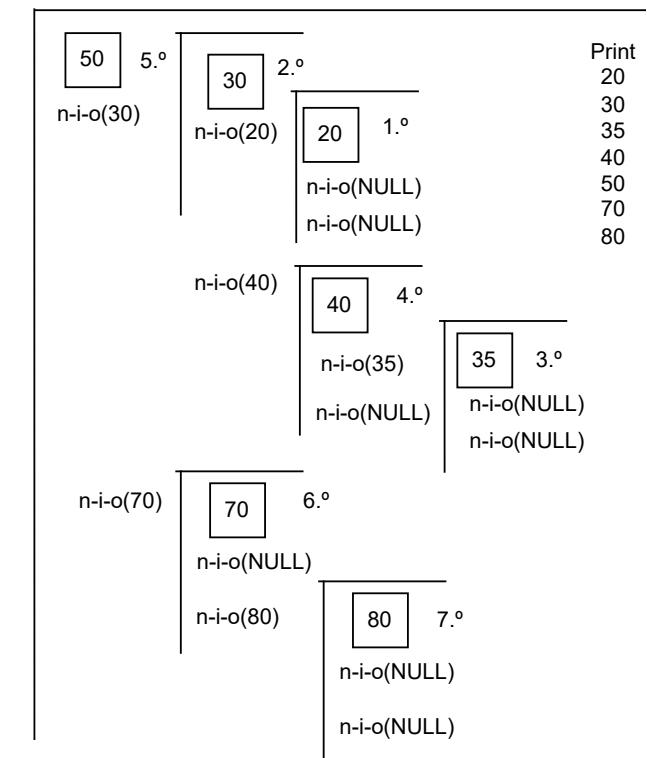
void mostrar_inorder(ptARVORE A) {
    if (!A)
        printf("nao definida...");
    else if (!A->raiz)
        printf("Arvore vazia...");
    else
        mostrar_inorder_no(A->raiz);
}

/* 12a travessia inorder */
void mostrar_inorder_no(ptNO no_actual) {
    if (no_actual) {
        mostrar_inorder_no(no_actual->esq);
        mostrar_no(no_actual);
        printf("\n");
        mostrar_inorder_no(no_actual->dir);
    }
}
  
```

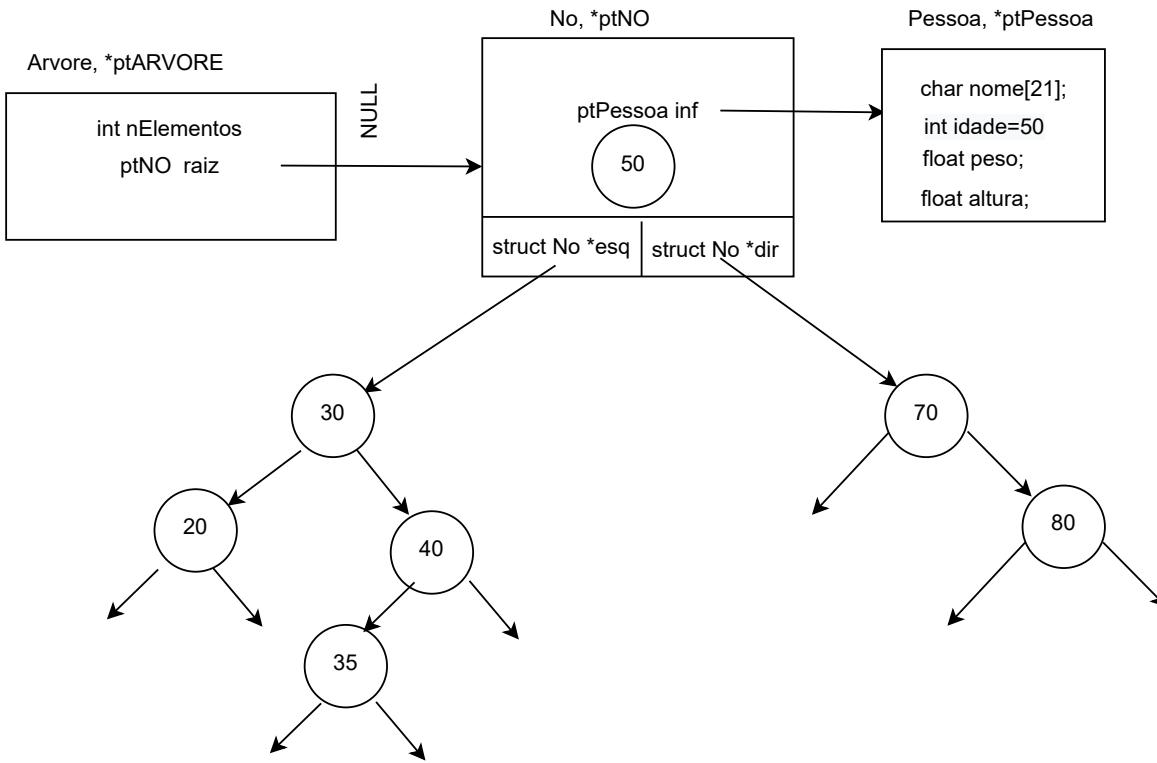
Listar todos os nós da árvore (travessia in order - por ordem de idades):
1.º visita o conteúdo da subárvore esquerda;
2.º em seguida o nó raiz;
3.º visita a subárvore direita

```

case 3: // travessia inorder
if (arv->nElementos==0)
    printf("\nNao existem nos na arvore!");
else
    mostrar_inorder(arv);
break; // _____
  
```



Mostrar todos os Nós da árvore usando a técnica: (4)preorder



```

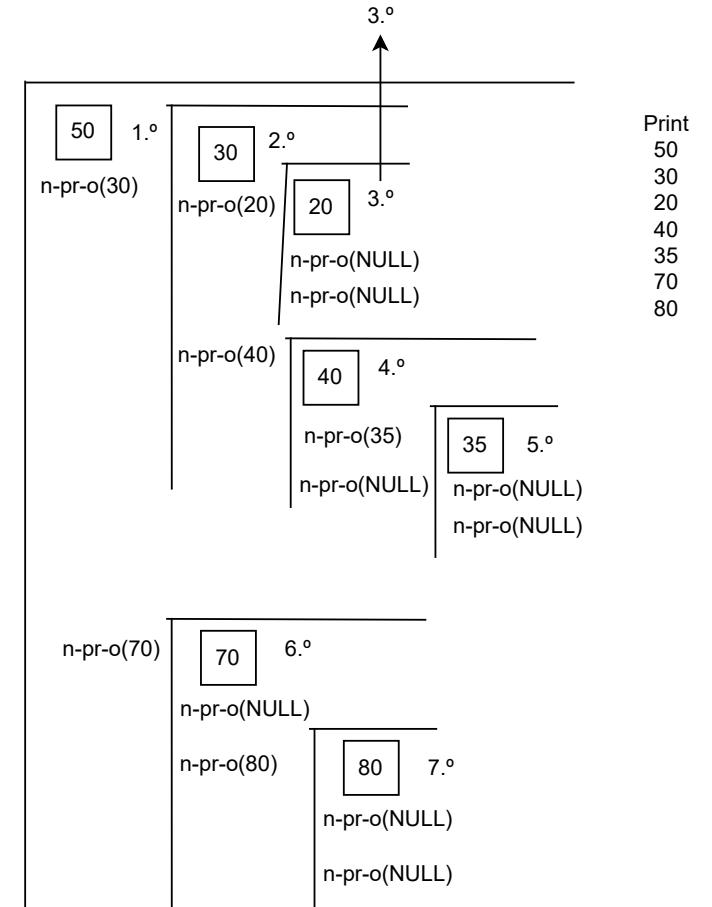
void mostrar_preorder(ptARVORE A) {
    if (!A)
        printf("Arvore nao definida...");
    else if (!A->raiz)
        printf("Arvore vazia...");
    else
        mostrar_preorder_no(A->raiz);
}

/* 12b travessia preorder */
void mostrar_preorder_no (ptNO no_actual) {
    if (no_actual) {
        mostrar_no(no_actual);
        printf("\n");
        mostrar_preorder_no(no_actual->esq);
        mostrar_preorder_no(no_actual->dir);
    }
}
  
```

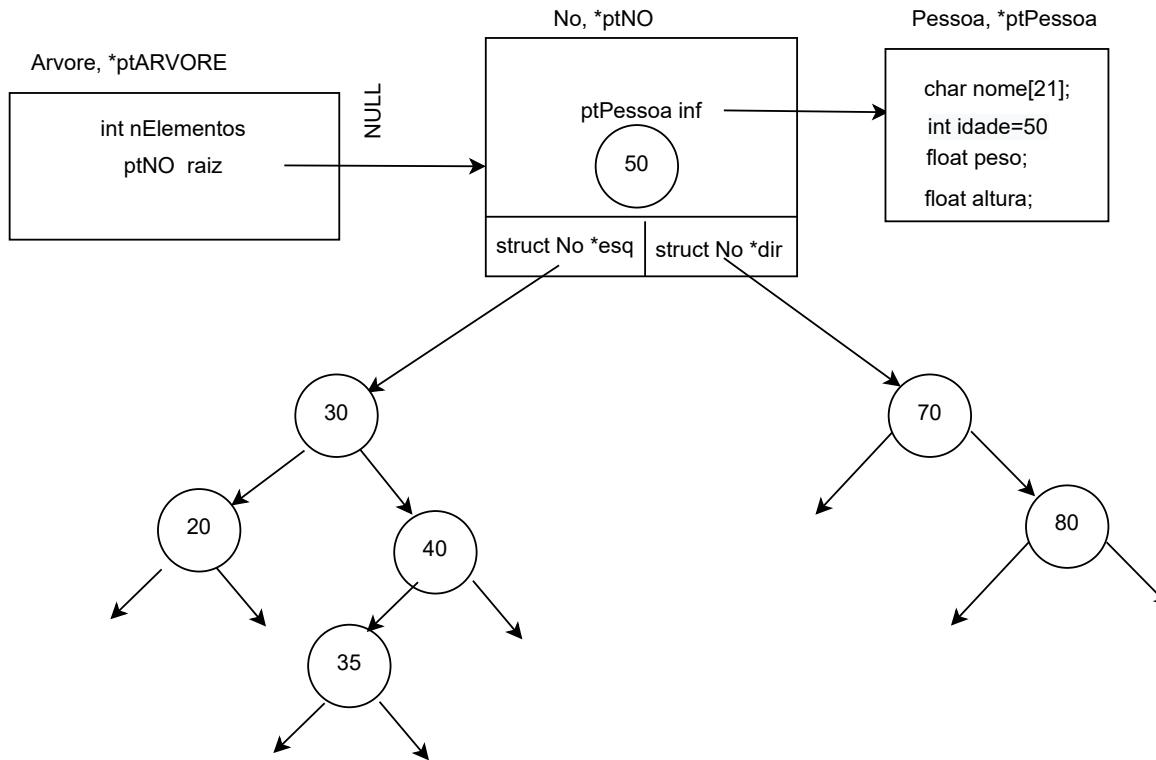
Listar todos os nós da árvore (travessia pre-order):
 1.º visita primeiro o nó raiz;
 2.º em seguida, o conteúdo da subárvore esquerda;
 3.º, depois a subárvore direita.

```

case 4: // travessia preorder
    if (arv->nElementos==0)
        printf("\nNao existem nos na arvore!");
    else
        mostrar_preorder(arv);
    break; //_
  
```



Mostrar todos os Nós da árvore usando a técnica:(5) postorder

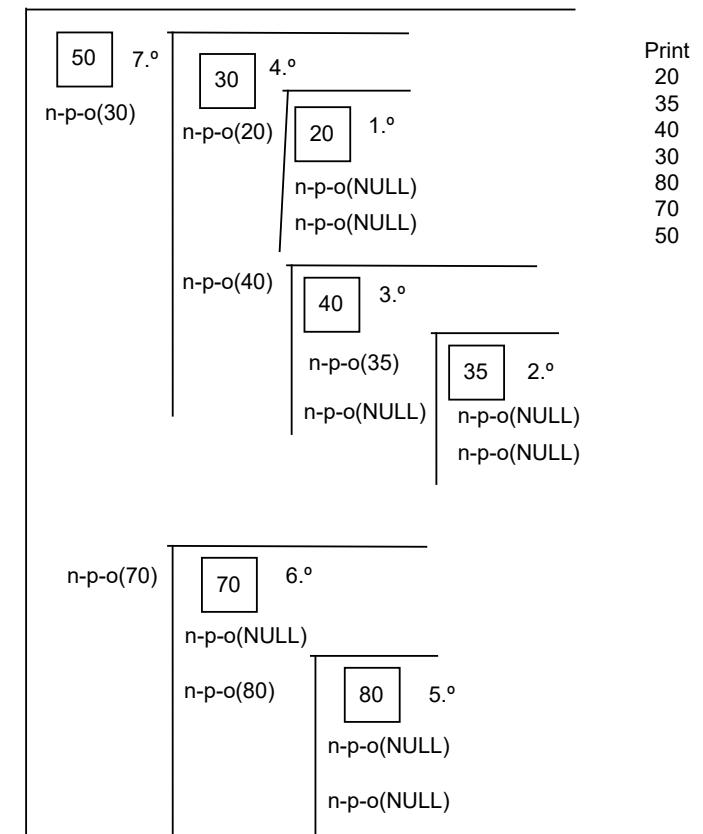


```
void mostrar_postorder(ptARVORE A) {
    if (!A)
        printf("Arvore nao definida...");
    else if (!A->raiz)
        printf("Arvore vazia...");
    else
        mostrar_postorder_no(A->raiz);
}
```

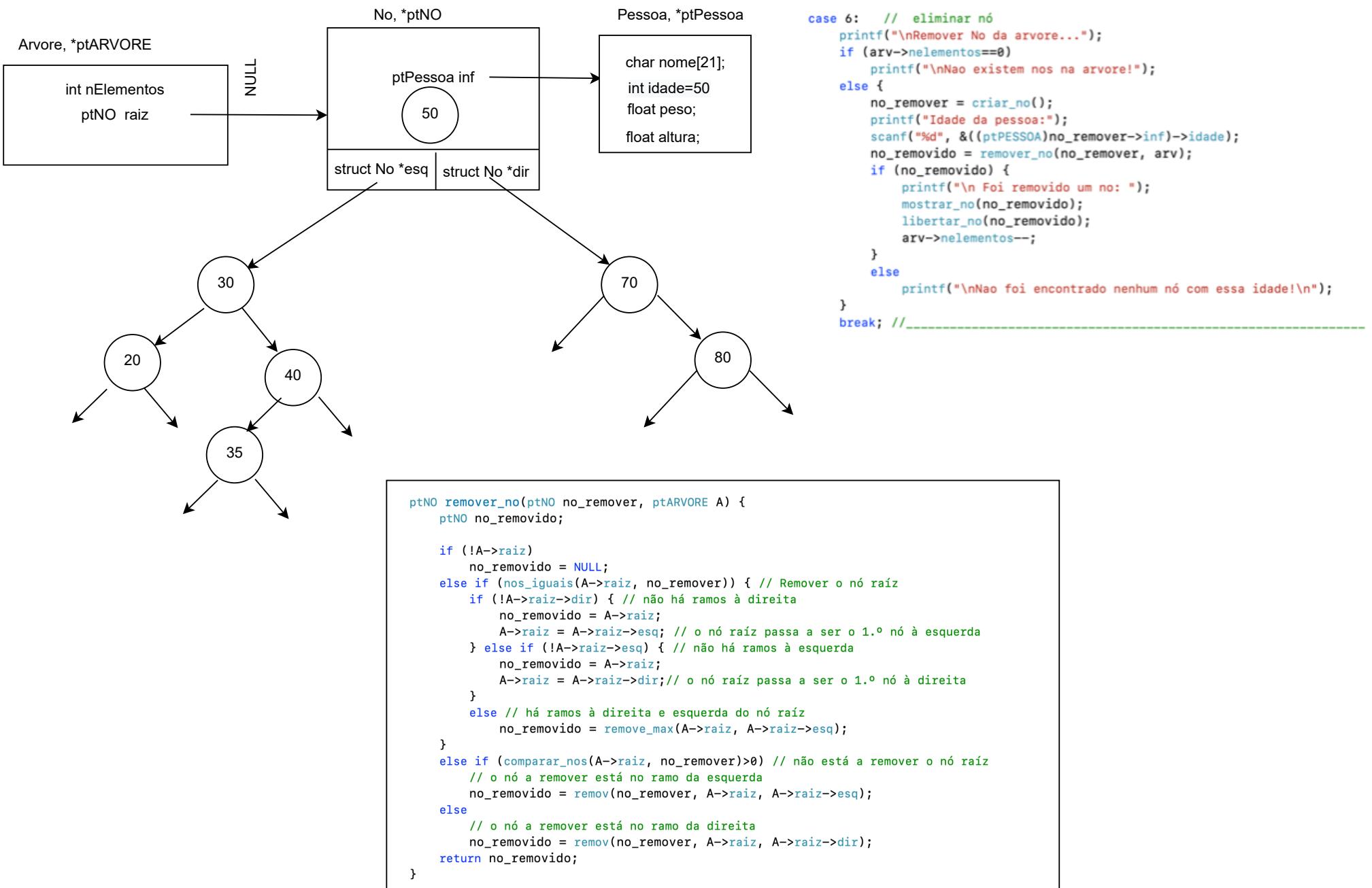
```
/* 12c travessia postorder */
void mostrar_postorder_no (ptNO no_actual) {
    if (no_actual) {
        mostrar_postorder_no(no_actual->esq);
        mostrar_postorder_no(no_actual->dir);
        mostrar_no(no_actual);
        printf("\n");
    }
}
```

Listar todos os nós da árvore (traversia pos-order):
1.^o visita primeiro o conteúdo da subárvore esquerda;
2.^o depois a subárvore direita;
3.^o no final, o nó raiz.

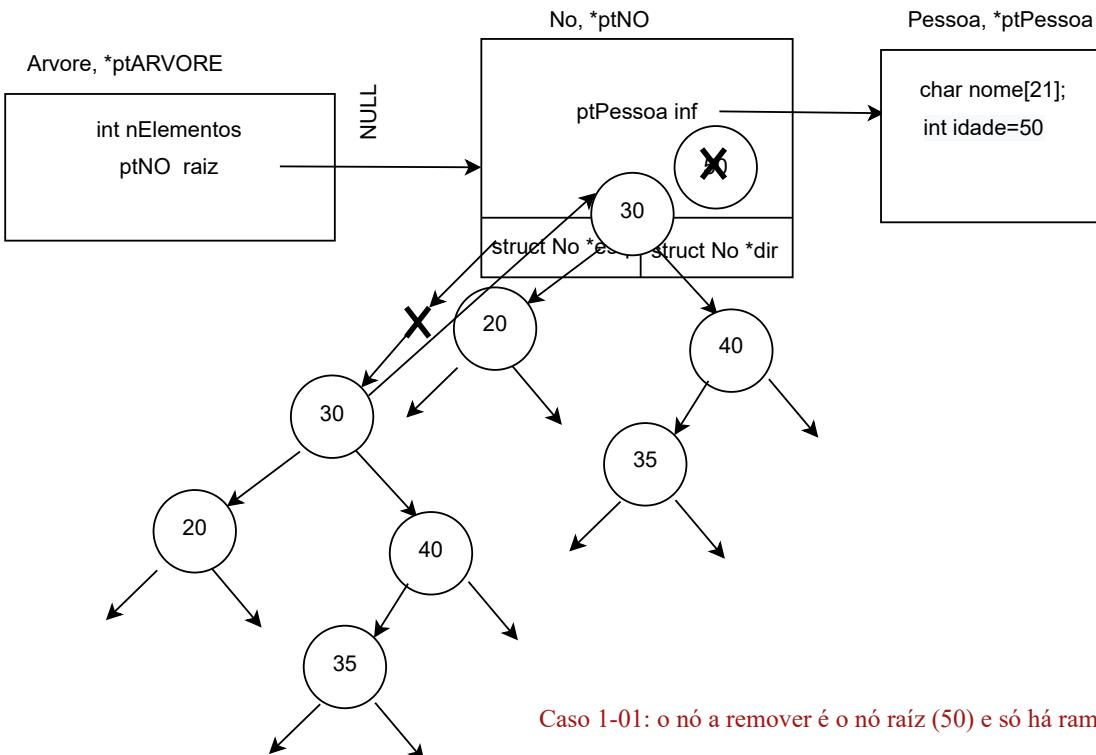
```
case 5: // travessia postorder
    if (arv->nElementos==0)
        printf("\nNao existem nos na arvore!");
    else
        mostrar_postorder(arv);
    break; //
```



(6) Retirar um Nó da árvore (através da idade)



(6) Retirar um Nó da árvore (através da idade)



```

case 6: // eliminar nó
printf("\nRemover No da arvore...");
if (arv->nElementos==0)
    printf("\nNao existem nos na arvore!");
else {
    no_remover = criar_no();
    printf("Idade da pessoa:");
    scanf("%d", &((ptPESSOA)no_remover->inf)->idade);
    no_removido = remover_no(no_remover, arv);
    if (no_removido) {
        printf("\n Foi removido um no: ");
        mostrar_no(no_removido);
        libertar_no(no_removido);
        arv->nElementos--;
    }
    else
        printf("\nNao foi encontrado nenhum nó com essa idade!\n");
}
break; //_

```

Caso 1-01: o nó a remover é o nó raiz (50) e só há ramo à esquerda.

```

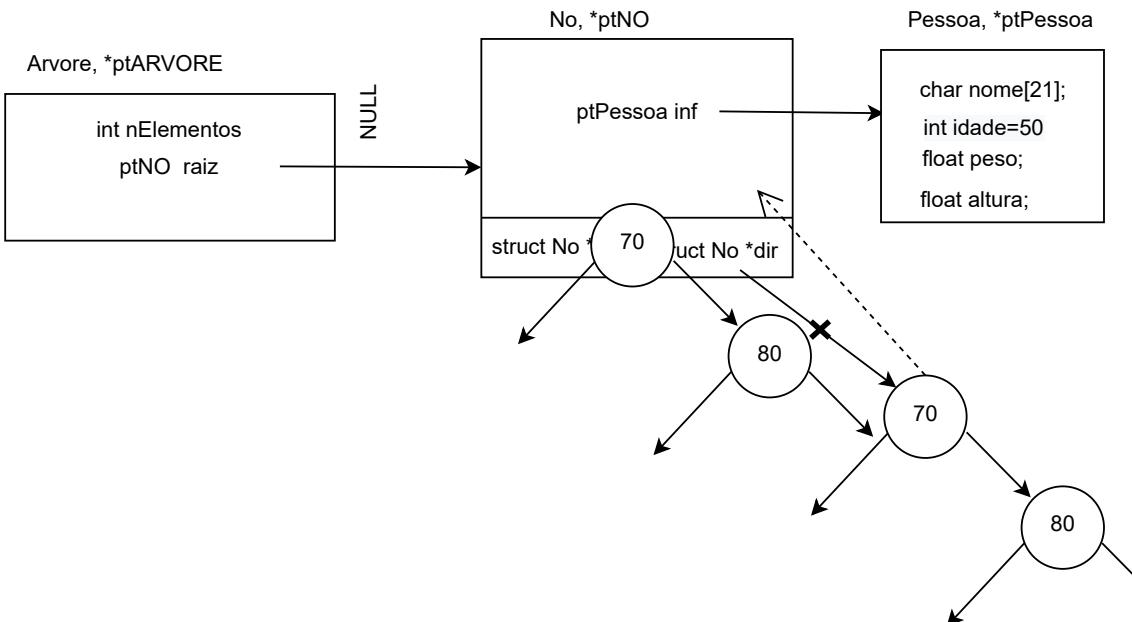
ptNO remover_no(ptNO no_remover, ptARVORE A) {
    ptNO no_removido;

    if (!A->raiz)
        no_removido = NULL;
    else if (nos_iguais(A->raiz, no_remover)) { // Remover o nó raiz
        if (!A->raiz->dir) { // não há ramos à direita
            no_removido = A->raiz;
            A->raiz = A->raiz->esq; // o nó raiz passa a ser o 1.º nó à esquerda
        } else if (!A->raiz->esq) { // não há ramos à esquerda
            no_removido = A->raiz;
            A->raiz = A->raiz->dir; // o nó raiz passa a ser o 1.º nó à direita
        }
        else // há ramos à direita e esquerda do nó raiz
            no_removido = remove_max(A->raiz, A->raiz->esq);
    }
}

```

A técnica é substituir o nó raiz pelo nó à esquerda, se o ramo à direita não existir e pelo nó à direita, se o ramo à esquerda não existir.

(6) Retirar um Nó da árvore (através da idade)



```

    case 6: // eliminar nó
        printf("\nRemover No da arvore...");
        if (arv->nElementos==0)
            printf("\nNao existem nos na arvore!");
        else {
            no_remover = criar_no();
            printf("Idade da pessoa:");
            scanf("%d", &((ptPESSOA)no_remover->inf)->idade);
            no_removido = remover_no(no_remover, arv);
            if (no_removido) {
                printf("\n Foi removido um no: ");
                mostrar_no(no_removido);
                libertar_no(no_removido);
                arv->nElementos--;
            }
            else
                printf("\nNao foi encontrado nenhum nó com essa idade!\n");
        }
        break; //_
    }
}

```

Caso 1-02: o nó a remover é o nó raiz (50) e só há ramo à direita.

```

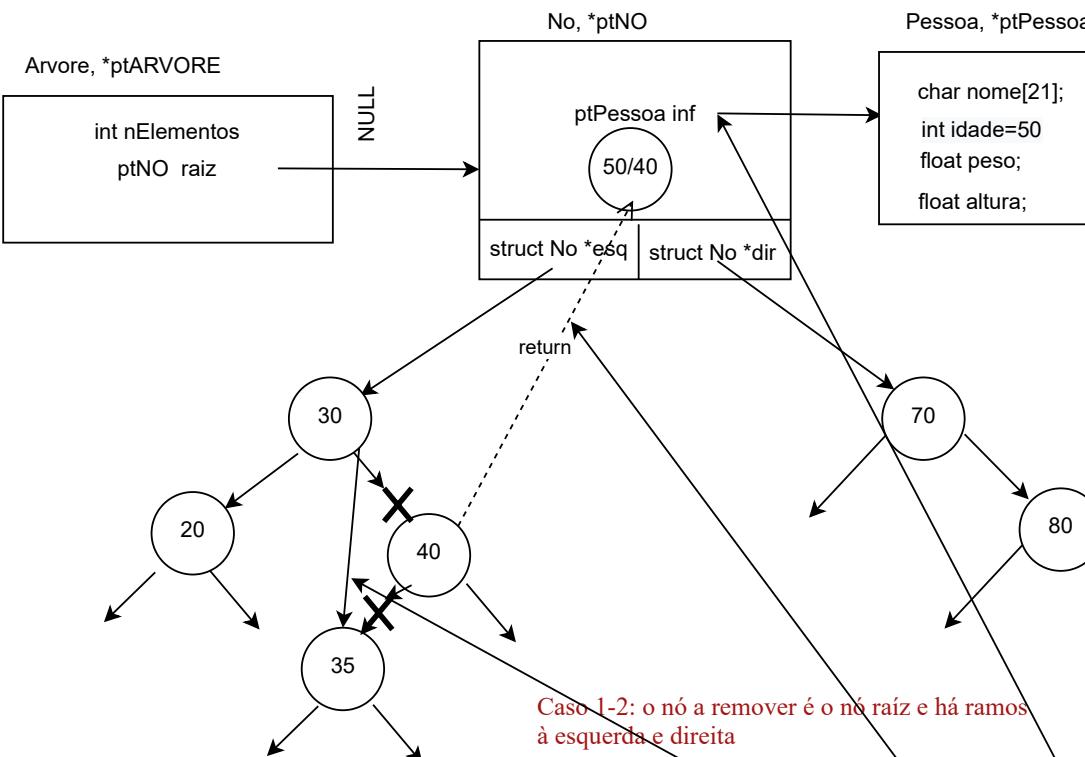
ptNO remover_no(ptNO no_remover, ptARVORE A) {
    ptNO no_removido;

    if (!A->raiz)
        no_removido = NULL;
    else if (nos_iguais(A->raiz, no_remover)) { // Remover o nó raiz
        if (!A->raiz->dir) { // não há ramos à direita
            no_removido = A->raiz;
            A->raiz = A->raiz->esq; // o nó raiz passa a ser o 1.º nó à esquerda
        } else if (!A->raiz->esq) { // não há ramos à esquerda
            no_removido = A->raiz;
            A->raiz = A->raiz->dir; // o nó raiz passa a ser o 1.º nó à direita
        }
        else // há ramos à direita e esquerda do nó raiz
            no_removido = remove_max(A->raiz, A->raiz->esq);
    }
}

```

A técnica é substituir o nó raiz pelo nó à esquerda, se o ramo à direita não existir e pelo nó à direita, se o ramo à esquerda não existir.

(6) Retirar um Nó da árvore (através da idade)



```

case 6: // eliminar nó
printf("\nRemover No da arvore...");
if (arv->nElementos==0)
    printf("\nNao existem nos na arvore!");
else {
    no_remover = criar_no();
    printf("Idade da pessoa:");
    scanf("%d", &((ptPESSOA)no_remover->inf)->idade);
    no_removido = remover_no(no_remover, arv);
    if (no_removido) {
        printf("\n Foi removido um no: ");
        mostrar_no(no_removido);
        libertar_no(no_removido);
        arv->nElementos--;
    }
    else
        printf("\nNao foi encontrado nenhum nó com essa idade!\n");
}
break; //_
  
```

```

ptNO remover_no(ptNO no_remover, ptARVORE A) {
    ptNO no_removido;

    if (!A->raiz)
        no_removido = NULL;
    else if (nos_iguais(A->raiz, no_remover)) { // Remover o nó raiz
        if (!A->raiz->dir) { // não há ramos à direita
            no_removido = A->raiz;
            A->raiz = A->raiz->esq; // o nó raiz passa a ser o 1.º nó à esquerda
        } else if (!A->raiz->esq) { // não há ramos à esquerda
            no_removido = A->raiz;
            A->raiz = A->raiz->dir; // o nó raiz passa a ser o 1.º nó à direita
        }
        else // há ramos à direita e esquerda do nó raiz
            no_removido = remove_max(A->raiz, A->raiz->esq);
    }
}
  
```

A técnica é substituir o nó pelo maior da sub-árvore que começa no filho à esquerda, neste caso, 50->30, o 30, ou seja o maior nó na sub-árvore que inicia em 30.

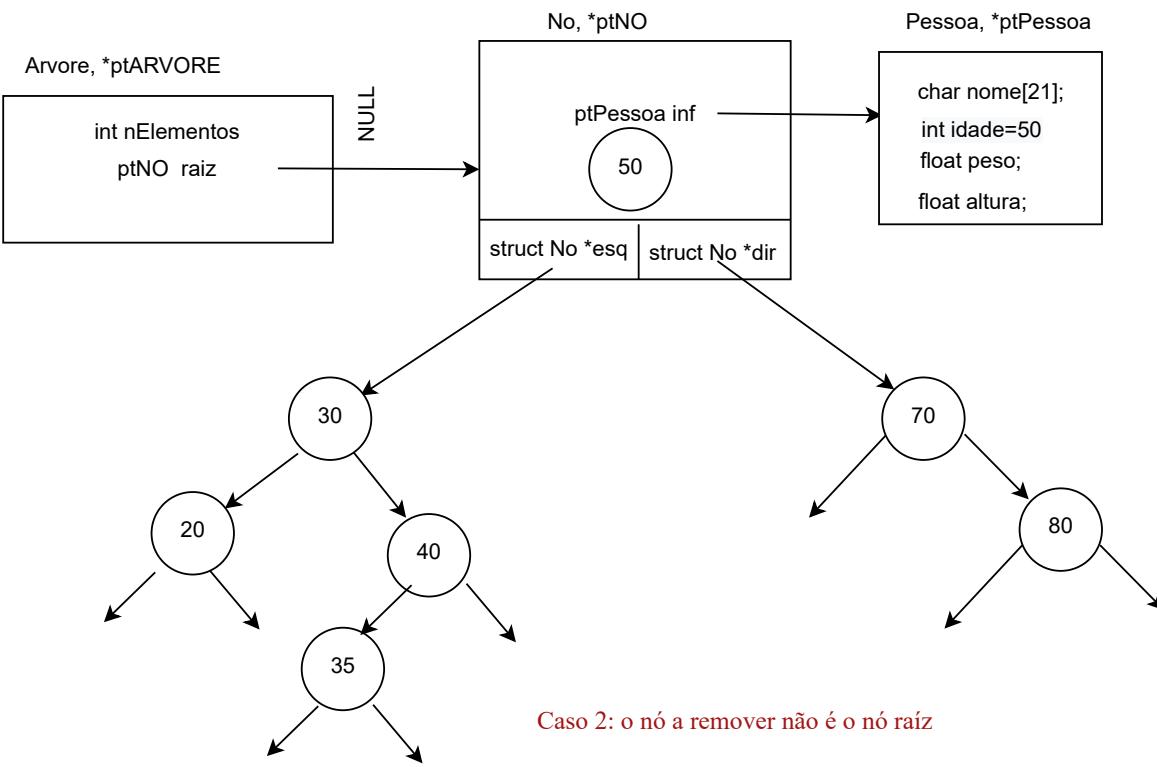
Começa-se no 30 e vai-se sempre para a direita e chega-se ao 40, que pode ter um filho à esquerda, ou nenhum (não pode ter um filho à direita, pois foi até ao final do caminho à direita).

Então, fazem-se as operações indicadas: o anterior (30) vai ligar à direita, o segundo, à esquerda (35). No fim, o último (40) passa ao raíz.

```

ptNO remove_max(ptNO tr, ptNO pos) {
    ptNO ant;
    ant = pos;
    while(pos->dir) { // vai procurar o maior nó
        ant = pos; // à esquerda do a eliminar
        pos = pos->dir;
    }
    if (tr->esq == pos)
        tr->esq = pos->esq;
    else
        ant->dir = pos->esq; // o ponteiro à direita
        // do nó anterior passa a apontar para o seg. actual
        // trocar a informação entre o nós tr e pos para que
        // em pos retorne a informação do nó eliminado
    ptPESSOA aux = tr->inf; // a inf. do nó a eliminar fica em ai
    tr->inf = pos->inf ; // a inf. do nó pos (40) passa para o :
    pos->inf = aux ; // a inf. anterior do nó raiz (50) passa
    return pos; // retorna o nó raiz eliminado
}
  
```


(6) Retirar um Nó da árvore (através da idade)



```

ptNO_remover_no(ptNO no_remover, ptARVORE A) {
    ptNO no_removido;

    if (!A->raiz)
        no_removido = NULL;
    else if (nos_iguais(A->raiz, no_remover)) { // Remover o nó raiz
        if (!A->raiz->dir) { // não há ramos à direita
            no_removido = A->raiz;
            A->raiz = A->raiz->esq; // o nó raiz passa a ser o 1.º nó à esquerda
        } else if (!A->raiz->esq) { // não há ramos à esquerda
            no_removido = A->raiz;
            A->raiz = A->raiz->dir; // o nó raiz passa a ser o 1.º nó à direita
        }
        else // há ramos à direita e esquerda do nó raiz
            no_removido = remove_max(A->raiz, A->raiz->esq);
    }
    else if (comparar_nos(A->raiz, no_remover)>0) // não está a remover o nó raiz
        // o nó a remover está no ramo da esquerda
        no_removido = remov(no_remover, A->raiz, A->raiz->esq);
    else
        // o nó a remover está no ramo da direita
        no_removido = remov(no_remover, A->raiz, A->raiz->dir);
    return no_removido;
}
  
```

```

case 6: // eliminar nó
printf("\nRemover No da arvore...");
if (arv->nElementos==0)
    printf("\nNao existem nos na arvore!");
else {
    no_remover = criar_no();
    printf("Idade da pessoa:");
    scanf("%d", &((ptPESSOA)no_remover->inf)->idade);
    no_removido = remover_no(no_remover, arv);
    if (no_removido) {
        printf("\n Foi removido um no: ");
        mostrar_no(no_removido);
        libertar_no(no_removido);
        arv->nElementos--;
    }
    else
        printf("\nNao foi encontrado nenhum nó com essa idade!\n");
}
break; //_
  
```

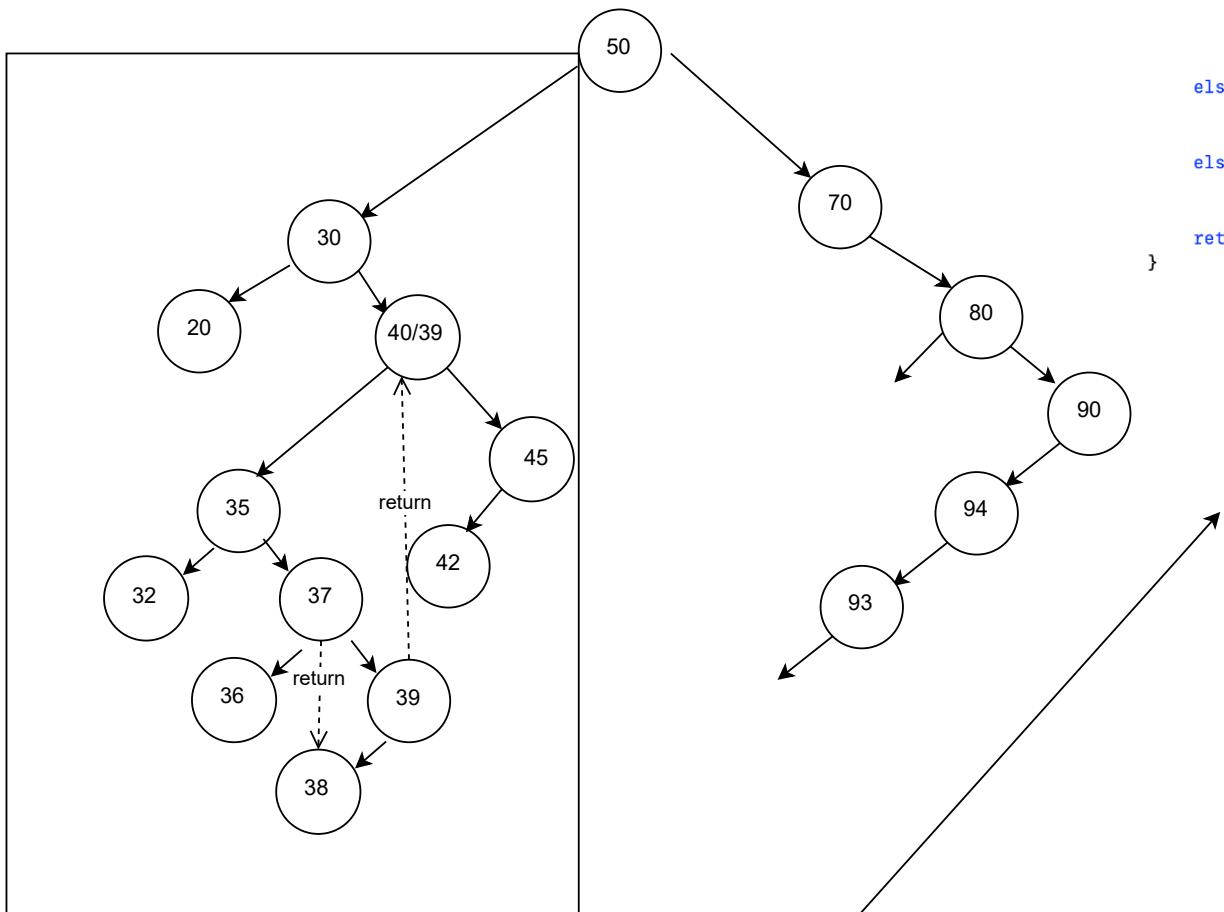
```

ptNO_remov(ptNO no_remover, ptNO ant, ptNO pos) {

    if (!pos)
        return NULL;
    else if (nos_iguais(pos, no_remover)) {
        if (!pos->dir) {
            if (ant->dir == pos)
                ant->dir=pos->esq;
            else
                ant->esq=pos->esq;
            return pos;
        }
        else if (!pos->esq) {
            if (ant->dir == pos)
                ant->dir=pos->dir;
            else
                ant->esq=pos->dir;
            return pos;
        }
        else
            return remove_max(pos,pos->esq);
    }
    else if (comparar_nos(pos, no_remover)>0)
        return remov(no_remover, pos, pos->esq);
    else
        return remov(no_remover, pos, pos->dir);
}
  
```

(6) Retirar um Nô da árvore (através da idade)

Caso 2.1: o nô a remover (40) nô é o nô raiz, e o nô a remover est à esquerda (ou seja o nô raiz é maior do que o nô a remover)



```
else if (comparar_nos(A->raiz, no_remover)>0) // nô est à a remover o nô raiz
    // o nô a remover est à no ramo da esquerda
    no_removido = remov(no_remover, A->raiz, A->raiz->esq);
    40      50      30
```

Solução:

O nô a remover (40) est à esquerda (ou seja, o nô a remover é maior do que o nô raiz).

Vamos eliminar o 40, que tem filhos à esquerda e à direita.

Chama-se a funcão Remov(...).

A tcnica é substituí-lo pelo maior nô da sub-árvore que comea no filho à esquerda do 40, ou seja, o maior nô que comea no 35 (vai-se sempre pela direita) e chega-se ao 39 (que pode ter 0 ou 1 filho à esquerda)

```
else if (comparar_nos(A->raiz, no_remover)>0) // nô est à a remover o nô raiz
    // o nô a remover est à no ramo da esquerda
    no_removido = remov(no_remover, A->raiz, A->raiz->esq);
else
    // o nô a remover est à no ramo da direita
    no_removido = remov(no_remover, A->raiz, A->raiz->dir);
return no_removido;
```

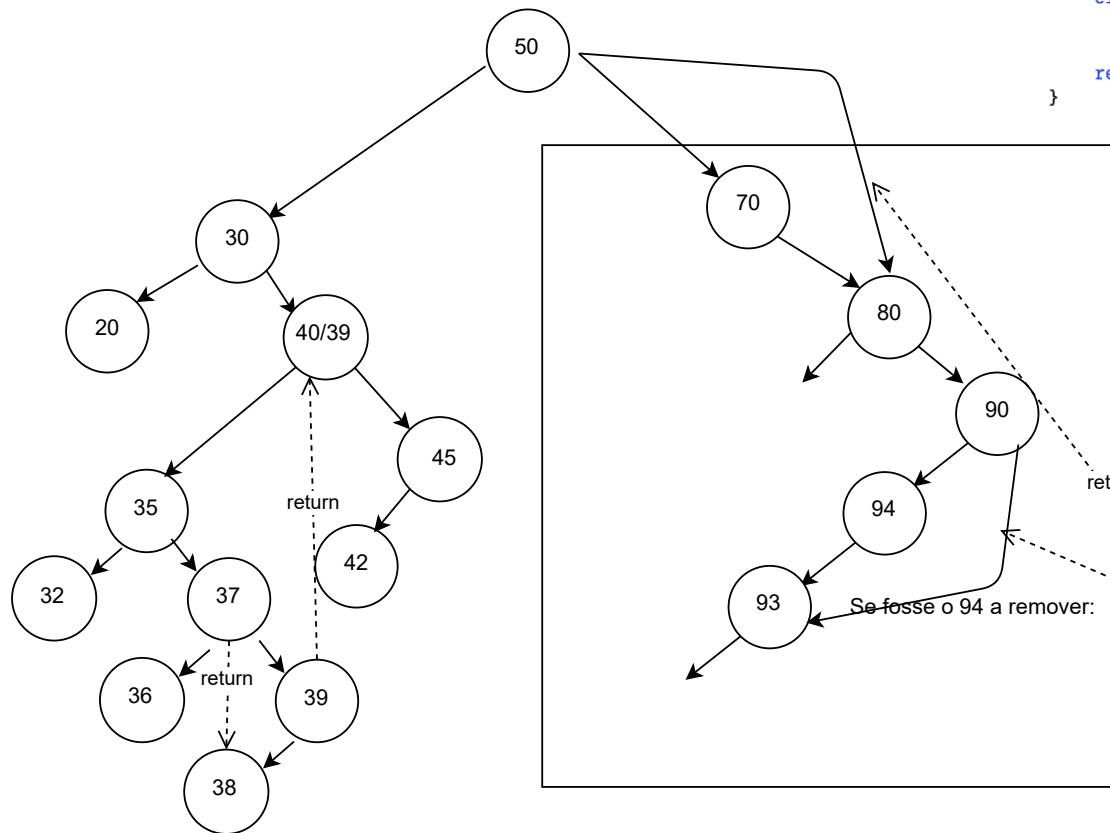
```
}
```

```
ptNO remov(ptNO no_remover, ptNO ant, ptNO pos) {
```

```
    if (!pos)
        return NULL;
    else if (nos_iguais(pos, no_remover)) {
        if (!pos->dir) {
            if (ant->dir == pos)
                ant->dir=pos->esq;
            else
                ant->esq=pos->esq;
            return pos;
        }
        else if (!pos->esq) {
            if (ant->dir == pos)
                ant->dir=pos->dir;
            else
                ant->esq=pos->dir;
            return pos;
        }
        else
            return remove_max(pos,pos->esq);
    }
    else if (comparar_nos(pos, no_remover)>0)
        return remov(no_remover, pos, pos->esq);
    else
        return remov(no_remover, pos, pos->dir);
```

(6) Retirar um Nô da árvore (através da idade)

Caso 2.2: o nô a remover (70) nô é o nô raiz, e o nô a remover est à direita (ou seja o nô raiz é menor do que o nô a remover)



```

else if (comparar_nos(A->raiz, no_remover)>0) // nô est à a remover o nô raiz
    // o nô a remover est à no ramo da esquerda
    no_removido = remov(no_remover, A->raiz, A->raiz->esq);
else
    // o nô a remover est à no ramo da direita
    no_removido = remov(no_remover, A->raiz, A->raiz->dir);
return no_removido;
}

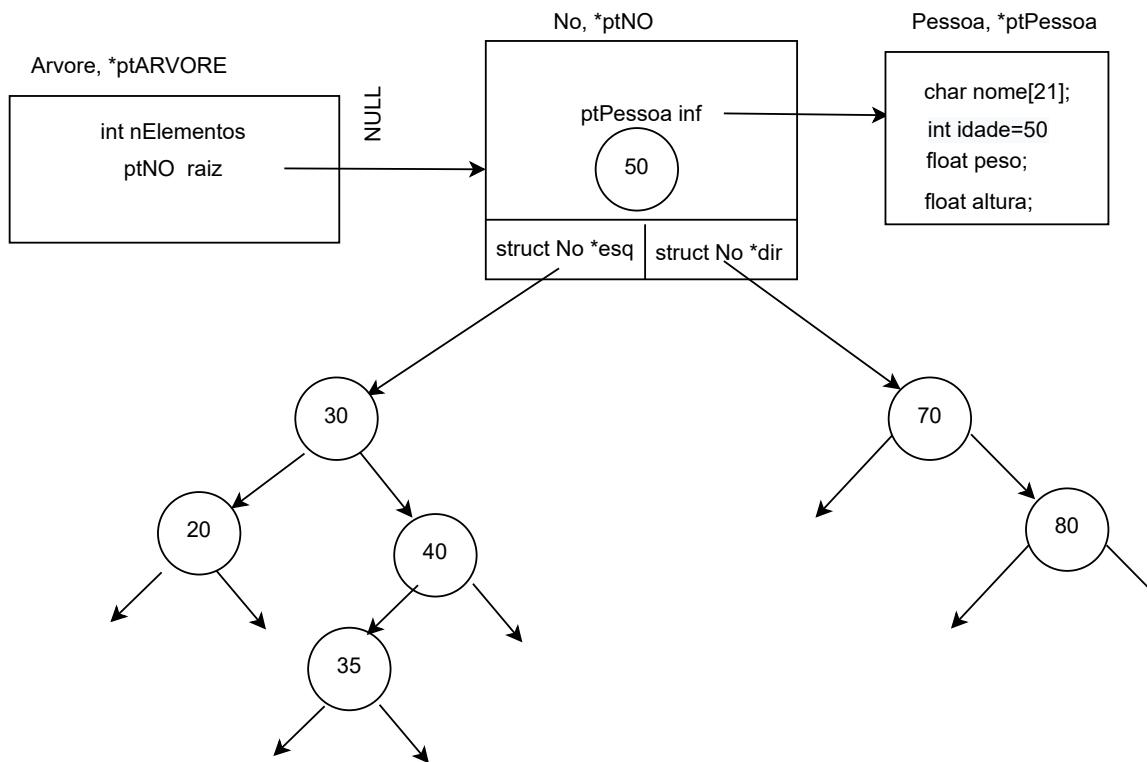
```

```

70      A->raiz      A->raiz->dir
ptNO remov(ptNO no_remover, ptNO ant, ptNO pos) {
    if (!pos)
        return NULL;
    else if (nos_iguais(pos, no_remover)) {
        if (!pos->dir) {
            if (ant->dir == pos)
                ant->dir=pos->esq;
            else
                ant->esq=pos->esq;
            return pos;
        }
        else if (!pos->esq) {
            if (ant->dir == pos)
                ant->dir=pos->dir;
            else
                ant->esq=pos->dir;
            return pos;
        }
        else
            return remove_max(pos,pos->esq);
    }
    else if (comparar_nos(pos, no_remover)>0)
        return remov(no_remover, pos, pos->esq);
    else
        return remov(no_remover, pos, pos->dir);
}

```

(7) Mostrar o número total de Nós da árvore



```

case 7:
    //printf("\nNumero total de nos:%d", arv->nelementos);
    printf("\nNumero total de nos:%d ", contar_nos(arv));
    printf("(arv->nElementos = %d)\n", arv->nElementos);
    break; //_

```

```

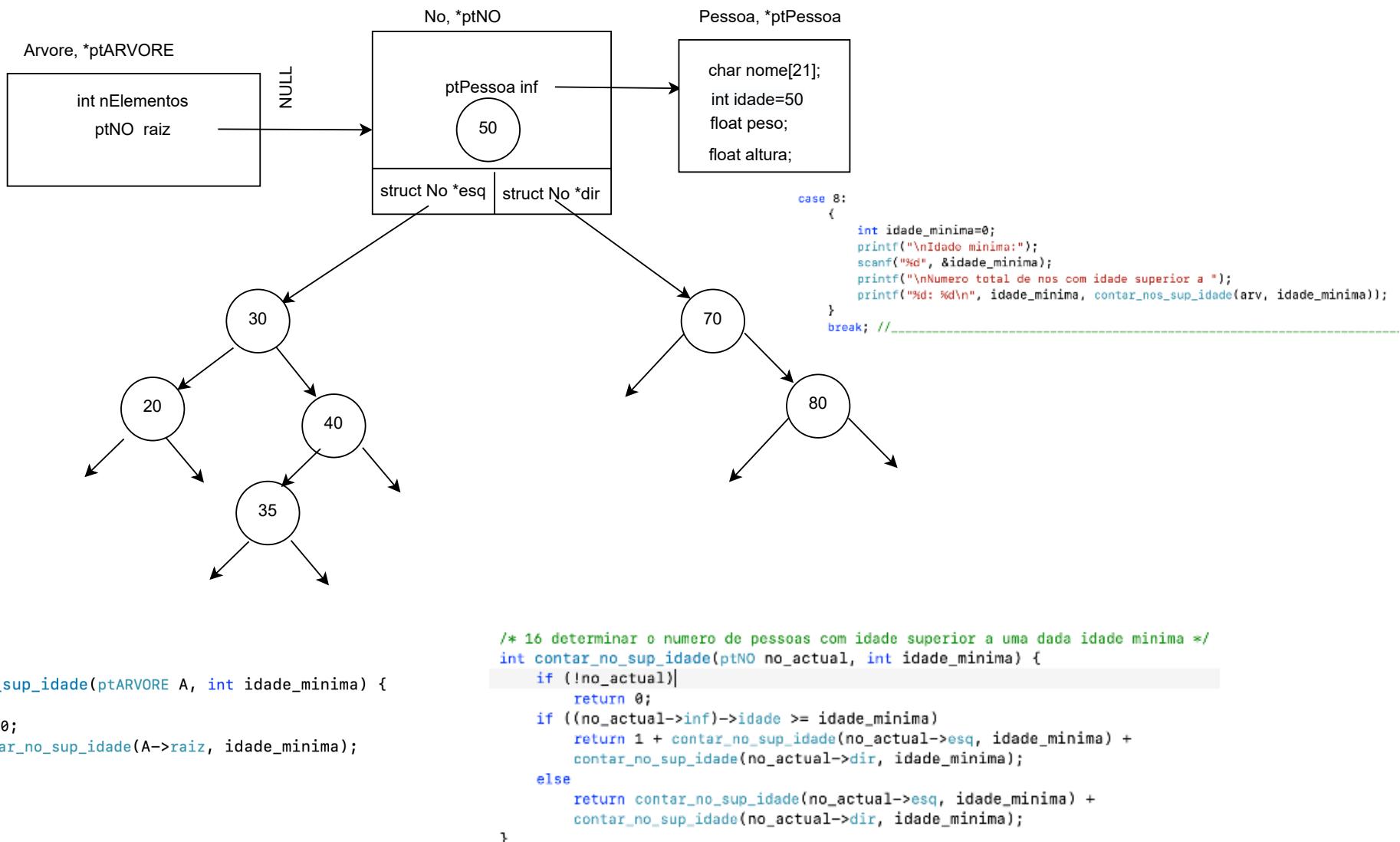
int contar_nos(ptARVORE A) {
    if (!A)
        return 0;
    return contar_no(A->raiz);
}

/* 15 contar os nos existentes numa dada arvore */
int contar_no(ptNO no_actual) {
    if (!no_actual)
        return 0;
    return 1 + contar_no(no_actual->esq) + contar_no(no_actual->dir);
}

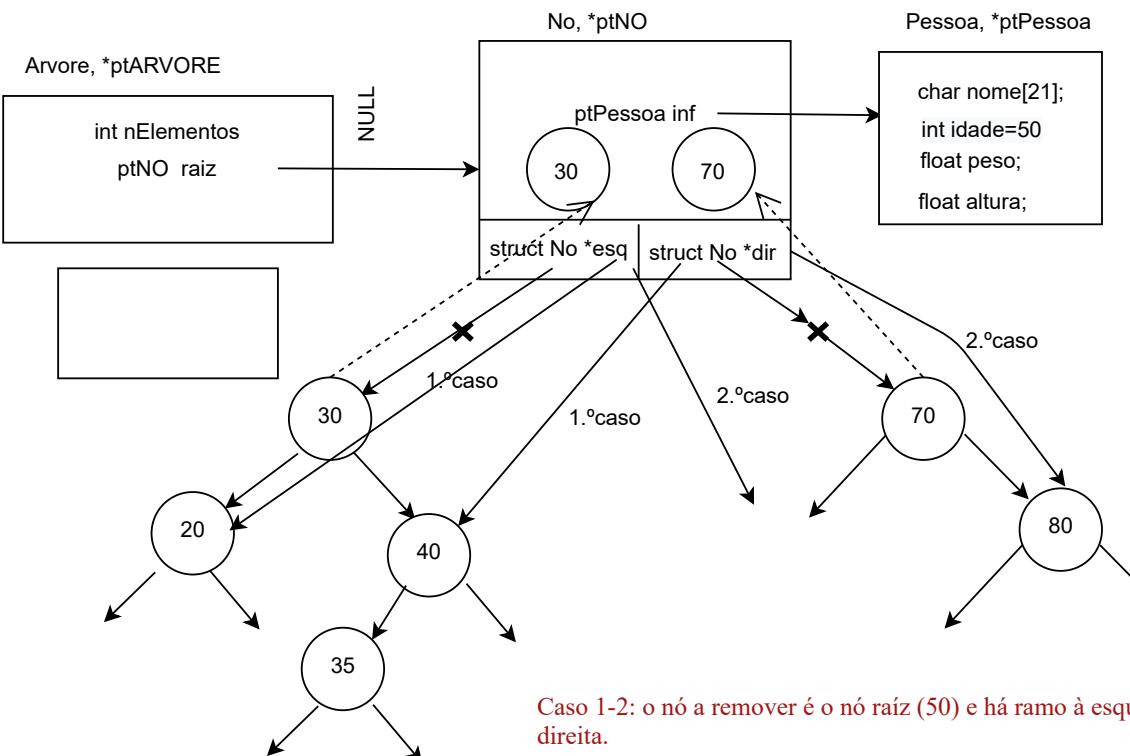
```

```
lementos);  
ios(arv));  
ementos);
```

(8) Mostrar o número de Nós na árvore com uma idade mínima



(6) Retirar um Nó da árvore (através da idade)



```

case 6: // eliminar nó
printf("\nRemover No da arvore...");
if (arv->nElementos==0)
  printf("\nNao existem nos na arvore!");
else {
  no_remover = criar_no();
  printf("Idade da pessoa:");
  scanf("%d", &((ptPESSOA)no_remover->inf)->idade);
  no_removido = remover_no(no_remover, arv);
  if (no_removido) {
    printf("\n Foi removido um no: ");
    mostrar_no(no_removido);
    libertar_no(no_removido);
    arv->nElementos--;
  }
  else
    printf("\nNao foi encontrado nenhum nó com essa idade!\n");
}
break; //_
  
```

Caso 1-2: o nó a remover é o nó raíz (50) e há ramo à esquerda ou direita.

```

ptNO remover_no(ptNO no_remover, ptARVORE A) {
  ptNO no_removido;

  if (!A->raiz)
    no_removido = NULL;
  else if (nos_iguais(A->raiz, no_remover)) { // Remover o nó raiz
    if (!A->raiz->dir) { // não há ramos à direita
      no_removido = A->raiz;
      A->raiz = A->raiz->esq; // o nó raiz passa a ser o 1.º nó à esquerda
    } else if (!A->raiz->esq) { // não há ramos à esquerda
      no_removido = A->raiz;
      A->raiz = A->raiz->dir; // o nó raiz passa a ser o 1.º nó à direita
    }
    else // há ramos à direita e esquerda do nó raiz
      no_removido = remove_max(A->raiz, A->raiz->esq);
  }
}
  
```

A técnica é substituir o nó pelo maior da sub-árvore que começa no filho à esquerda (neste caso o 30). Começa-se no 30 e vai-se sempre para a direita e chega-se ao 40, que pode ter um filho à esquerda ou nenhum (não pode ter um filho à direita, pois que se foi até ao fim do caminho à direita).

Então, fazem-se as operações indicadas:
o anterior vai ligar à direita, para o seg., à esquerda e depois, o nó último passa a ser o raiz.

```

ptNO remove_max(ptNO tr, ptNO pos) {
  ptNO ant;
  ant = pos;
  while(pos->dir) { // vai procurar o maior nó
    ant = pos; // à esquerda do a eliminar
    pos = pos->dir;
  }
  if (tr->esq == pos)
    tr->esq = pos->esq;
  else
    ant->dir = pos->esq; // o ponteiro à direita
    // do nó anterior passa a apontar para o seg. actual
    // trocar a informação entre o nós tr e pos para que
    // em pos retorne a informação do nó eliminado
  ptPESSOA aux = tr->inf;
  tr->inf = pos->inf; // a informação do nó pos
  pos->inf = aux; // passa para o raiz
  return pos;
}
  
```