



FAMEPI

Faculdade do Amazonas de Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

Algoritmos e Estrutura de Dados 2



Prof Pedro Corrêa



Objetivo da aula

Utilizar a Estrutura de Dados do tipo Árvore Binária de Pesquisa

Sumário

1. Generalidades

2. Árvores

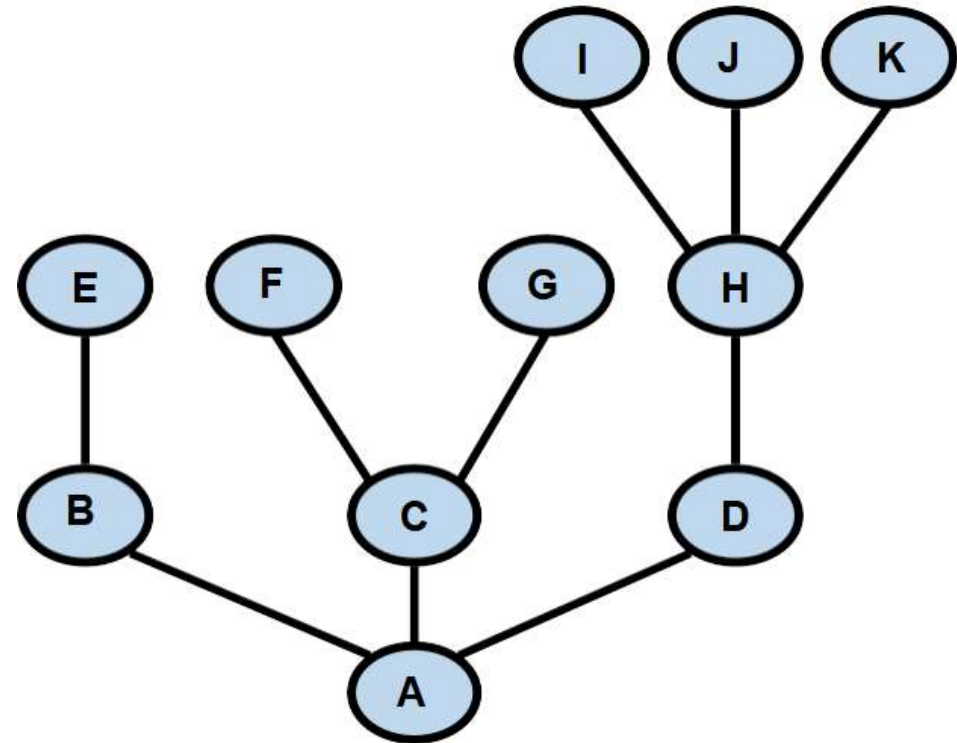
3. Árvore Binária de Pesquisa

4. Conclusão

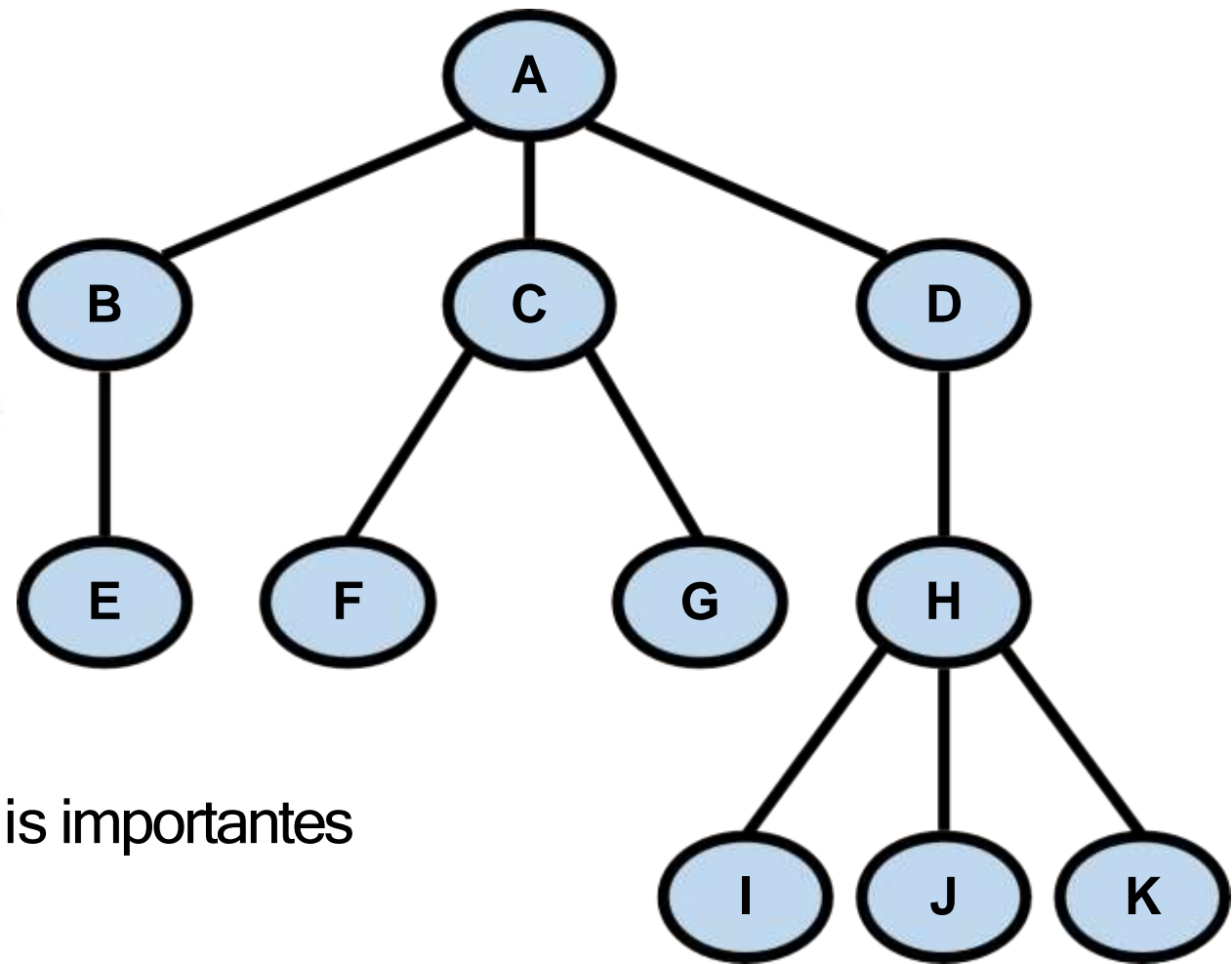
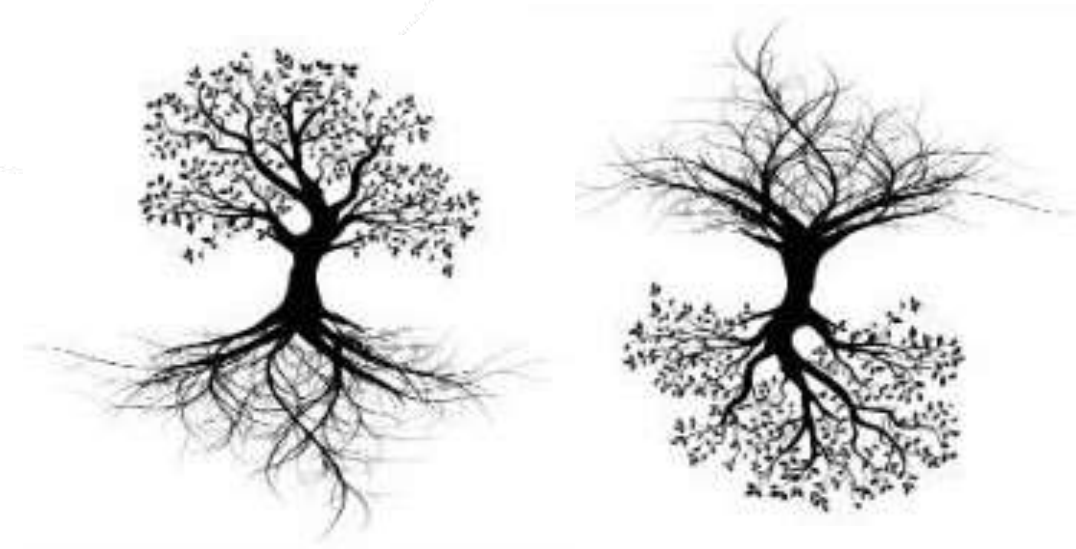


ÁRVORES

ÁRVORES



ÁRVORES

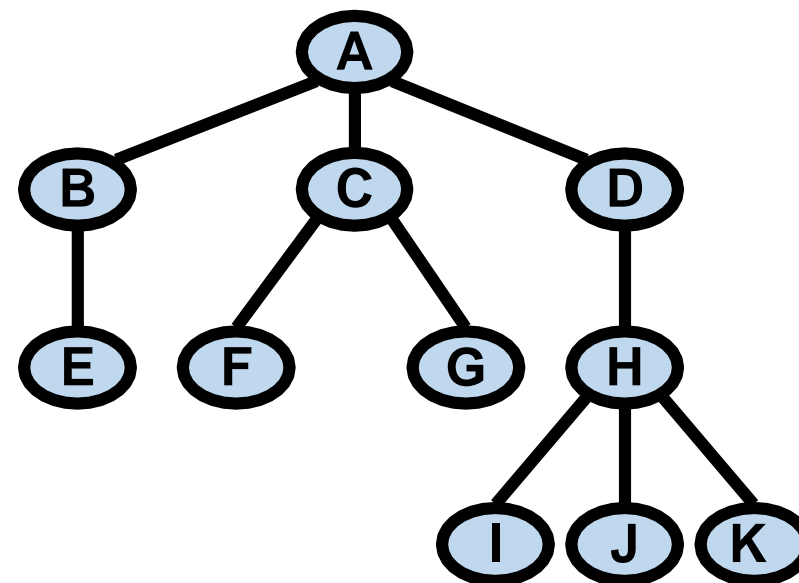


Constituem uma das estruturas mais importantes da área de computação.

São usadas em diversas aplicações

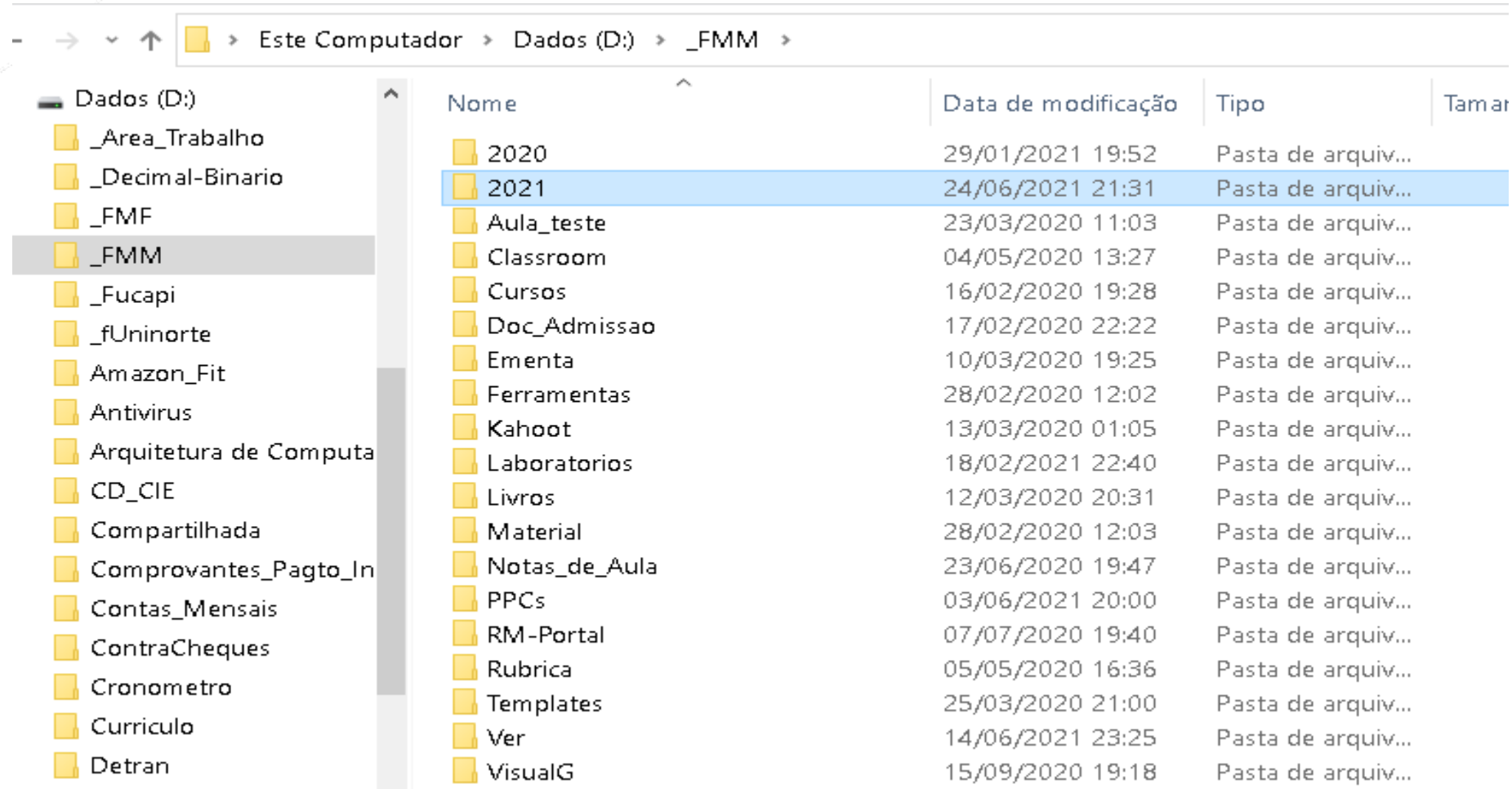
ÁRVORES

São formadas por um conjunto de nós que possuem relacionamento de **Hierarquia** ou **Subordinação**



EXEMPLOS DE APLICAÇÃO

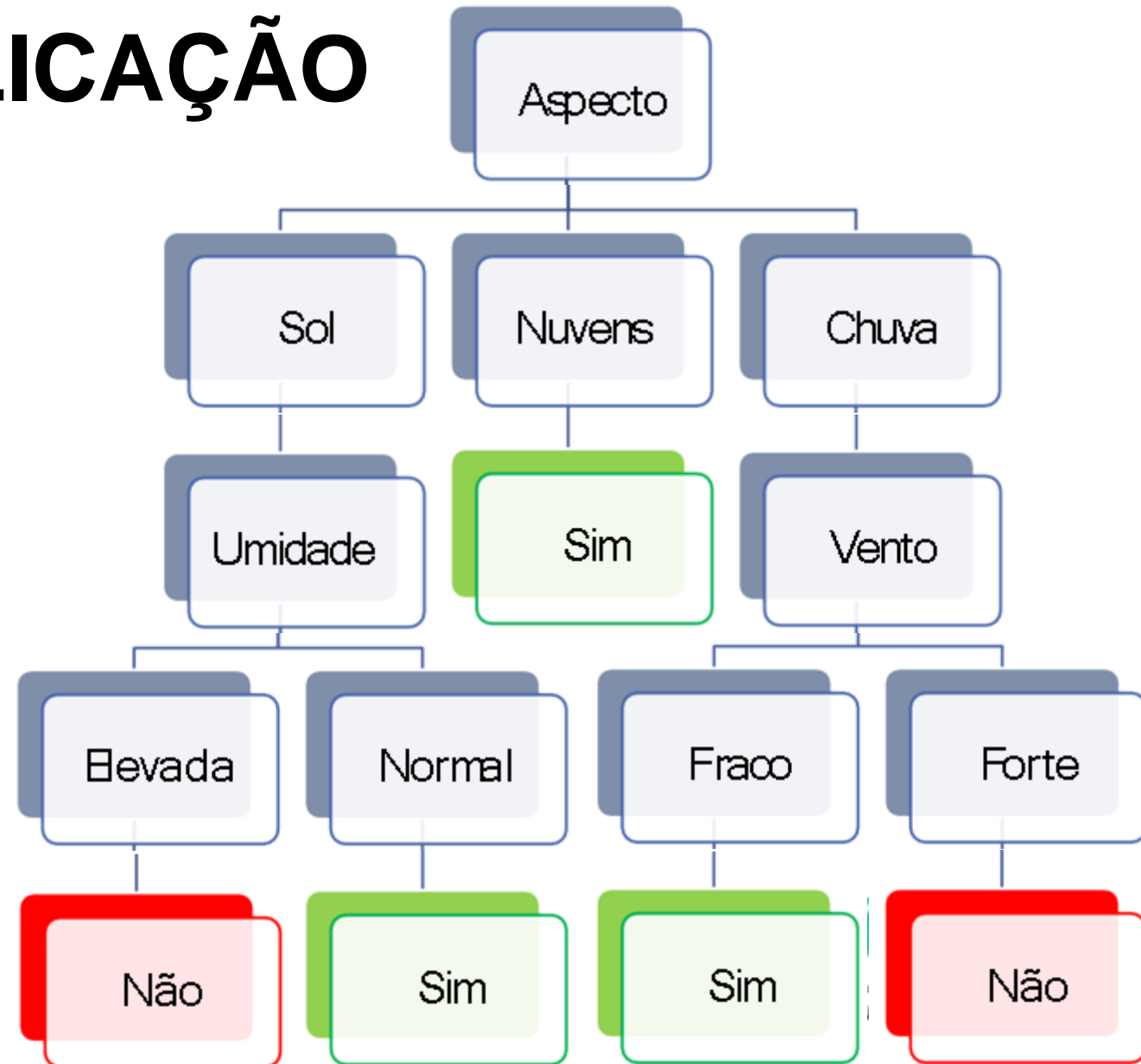
Sistema de arquivos



- > > > > > Este Computador > Dados (D:) > _FMM >				
Dados (D:)	Nome	Data de modificação	Tipo	Tam ar
_Area_Trabalho	2020	29/01/2021 19:52	Pasta de archiv...	
_Decimal-Binario	2021	24/06/2021 21:31	Pasta de archiv...	
_FMF	Aula_teste	23/03/2020 11:03	Pasta de archiv...	
_FMM	Classroom	04/05/2020 13:27	Pasta de archiv...	
_Fucapi	Cursos	16/02/2020 19:28	Pasta de archiv...	
_fUninorte	Doc_Admissao	17/02/2020 22:22	Pasta de archiv...	
Amazon_Fit	Ementa	10/03/2020 19:25	Pasta de archiv...	
Antivirus	Ferramentas	28/02/2020 12:02	Pasta de archiv...	
Arquitetura de Computa	Kahoot	13/03/2020 01:05	Pasta de archiv...	
CD_CIE	Laboratorios	18/02/2021 22:40	Pasta de archiv...	
Compartilhada	Livros	12/03/2020 20:31	Pasta de archiv...	
Comprovantes_Pagto_In	Material	28/02/2020 12:03	Pasta de archiv...	
Contas_Mensais	Notas_de_Aula	23/06/2020 19:47	Pasta de archiv...	
ContraCheques	PPCs	03/06/2021 20:00	Pasta de archiv...	
Cronometro	RM-Portal	07/07/2020 19:40	Pasta de archiv...	
Curriculo	Rubrica	05/05/2020 16:36	Pasta de archiv...	
Detran	Templates	25/03/2020 21:00	Pasta de archiv...	
	Ver	14/06/2021 23:25	Pasta de archiv...	
	VisualG	15/09/2020 19:18	Pasta de archiv...	

EXEMPLOS DE APLICAÇÃO

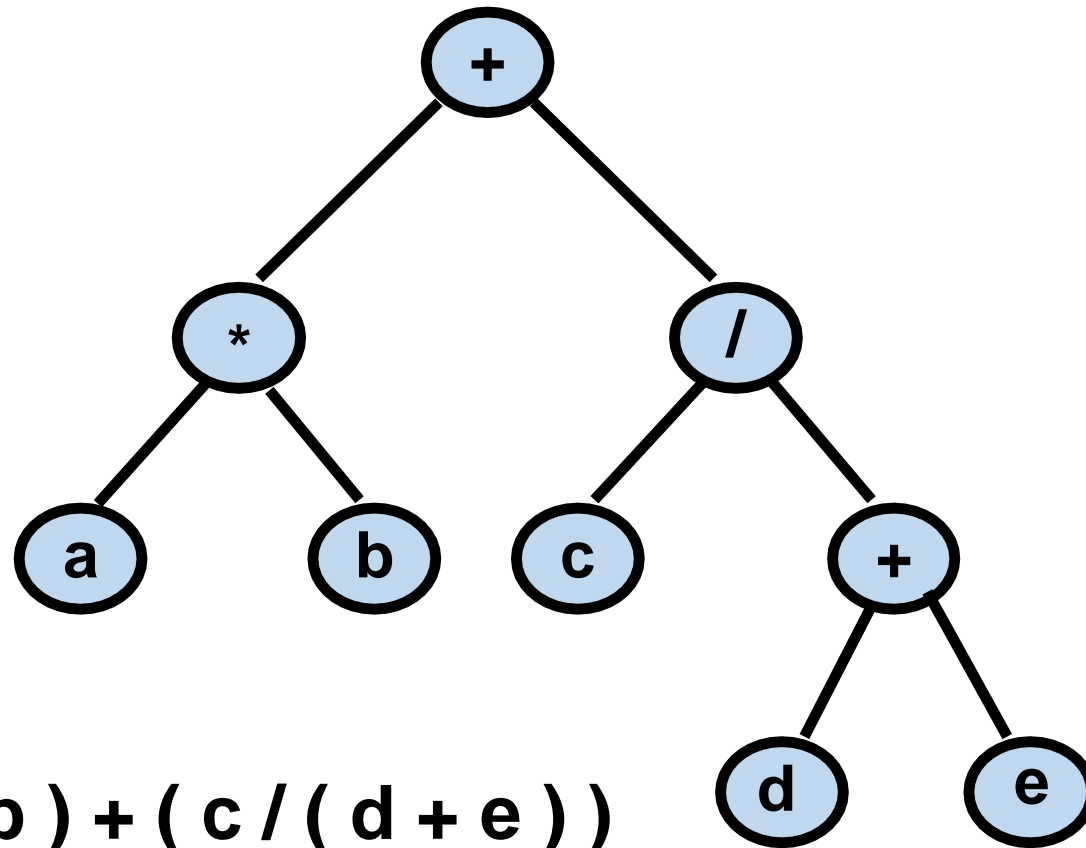
Árvore de decisão para jogar tênis



EXEMPLOS DE APLICAÇÃO

Árvore de derivação

- Usada pelos compiladores



Expressão aritmética: $(a * b) + (c / (d + e))$

DEFINIÇÃO FORMAL

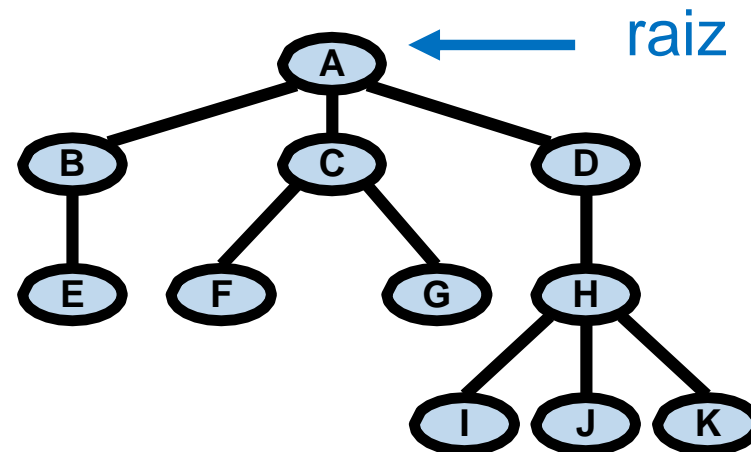
Conjunto finito de zero ou mais nós (**nodos** ou **vértices**), tal que:

Se o número de nós é maior do que zero:

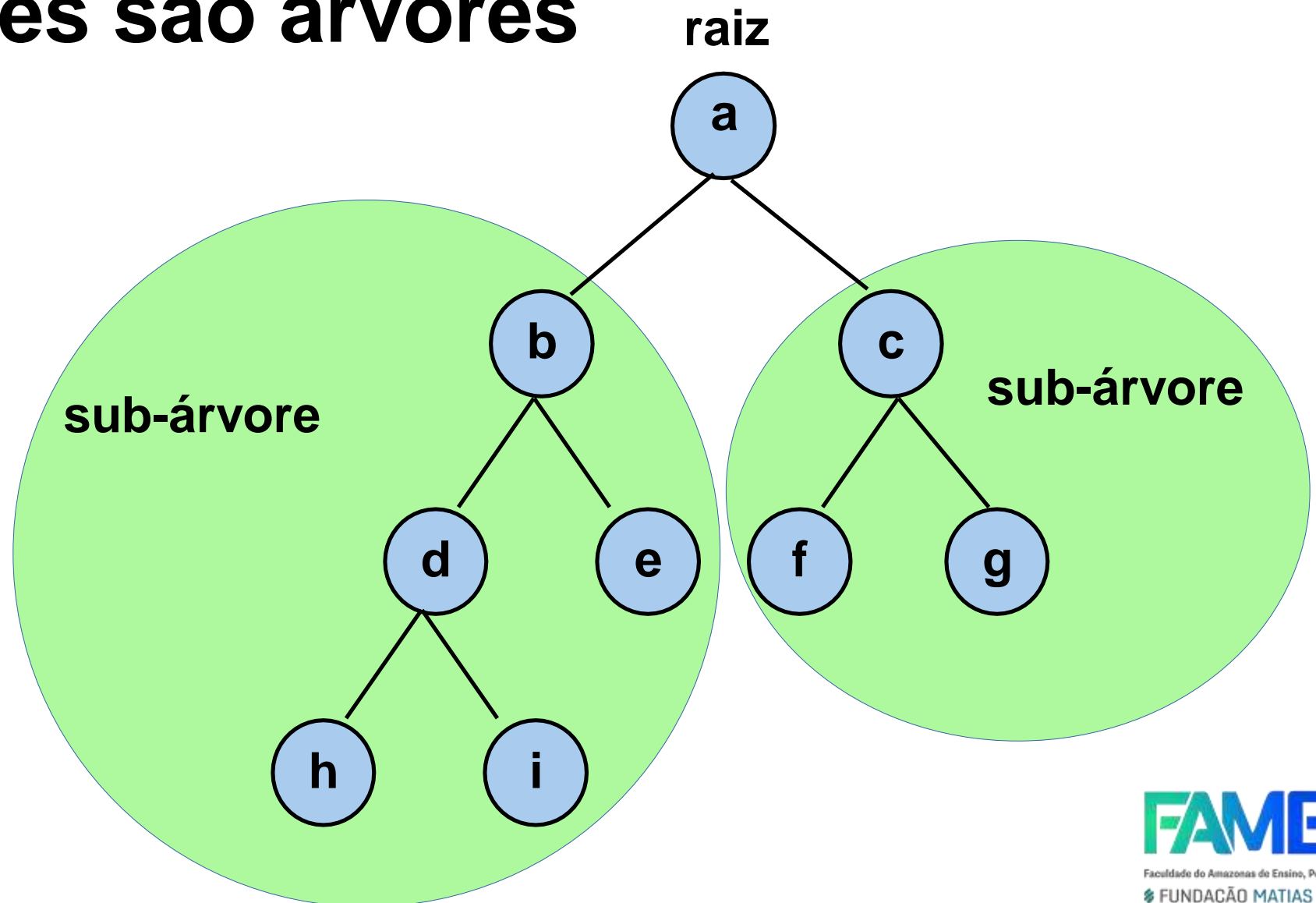
- Existe um nó denominado **raiz** da árvore
- Os demais nós formam $m \geq 0$ conjuntos disjuntos S_1, S_2, \dots, S_m , onde cada um destes é uma árvore (S_i são denominadas **sub-árvores**)

Se o número de nós é igual a zero:

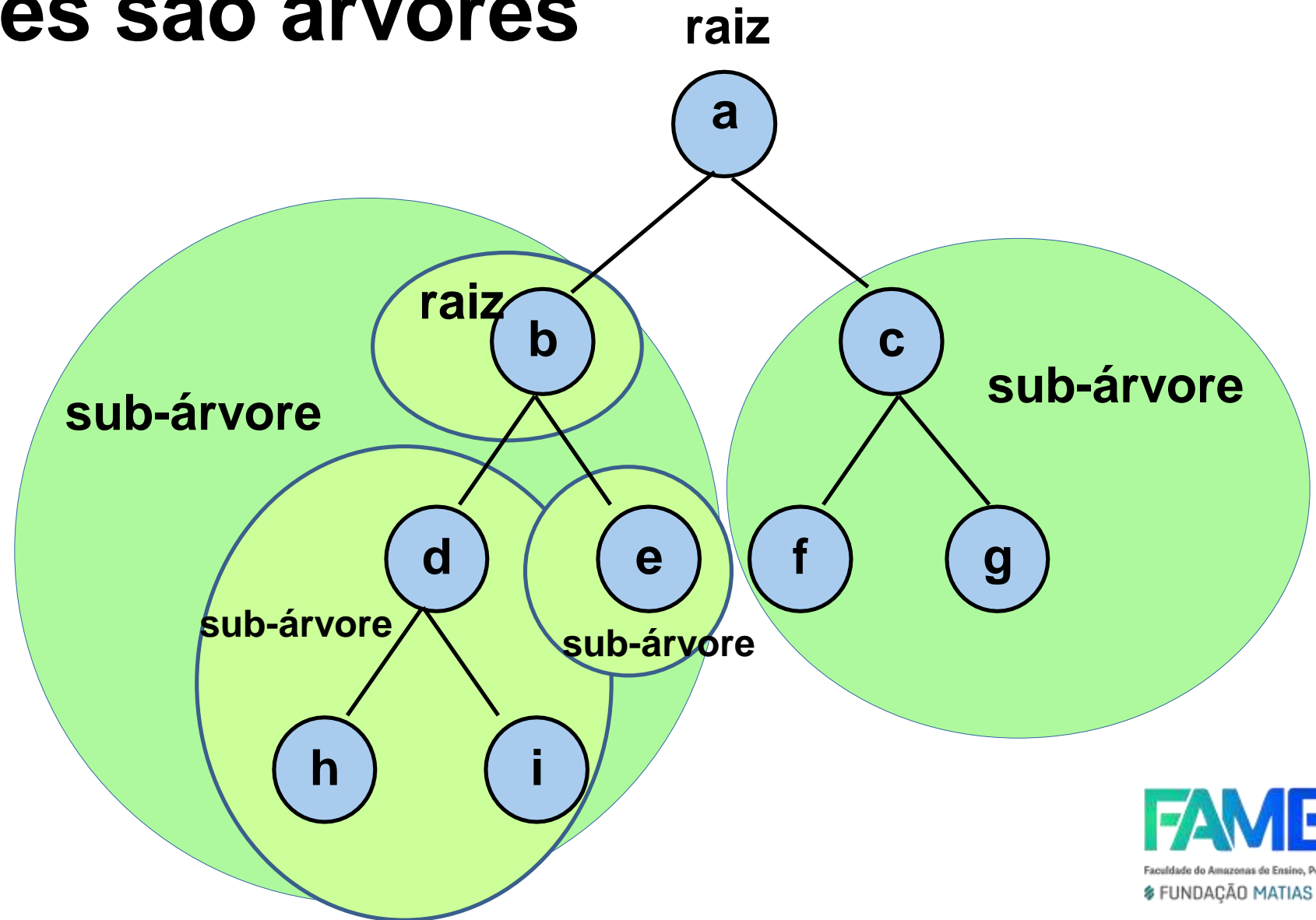
- **Árvore vazia**



Sub-árvores são árvores

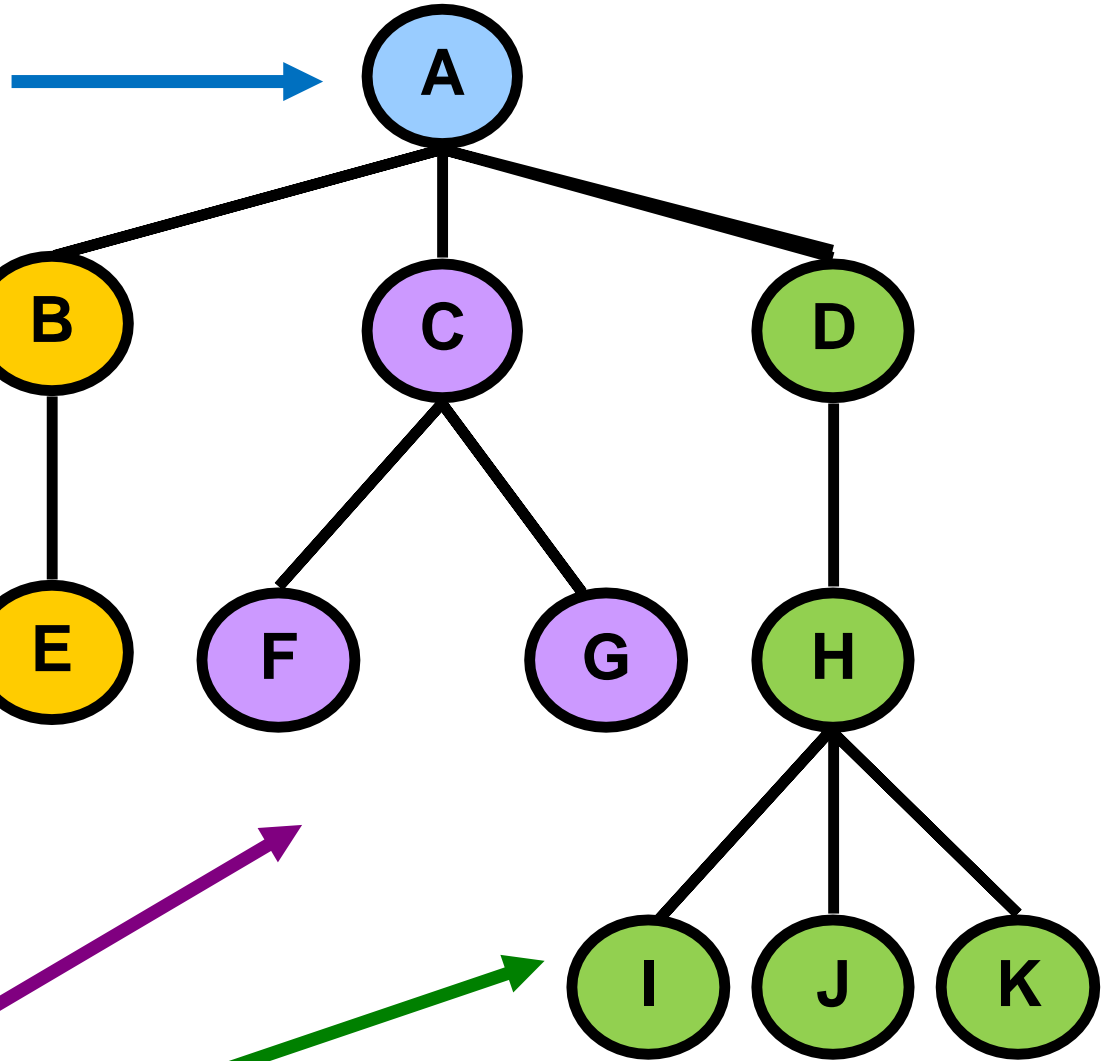


Sub-árvores são árvores



TERMINOLOGIA

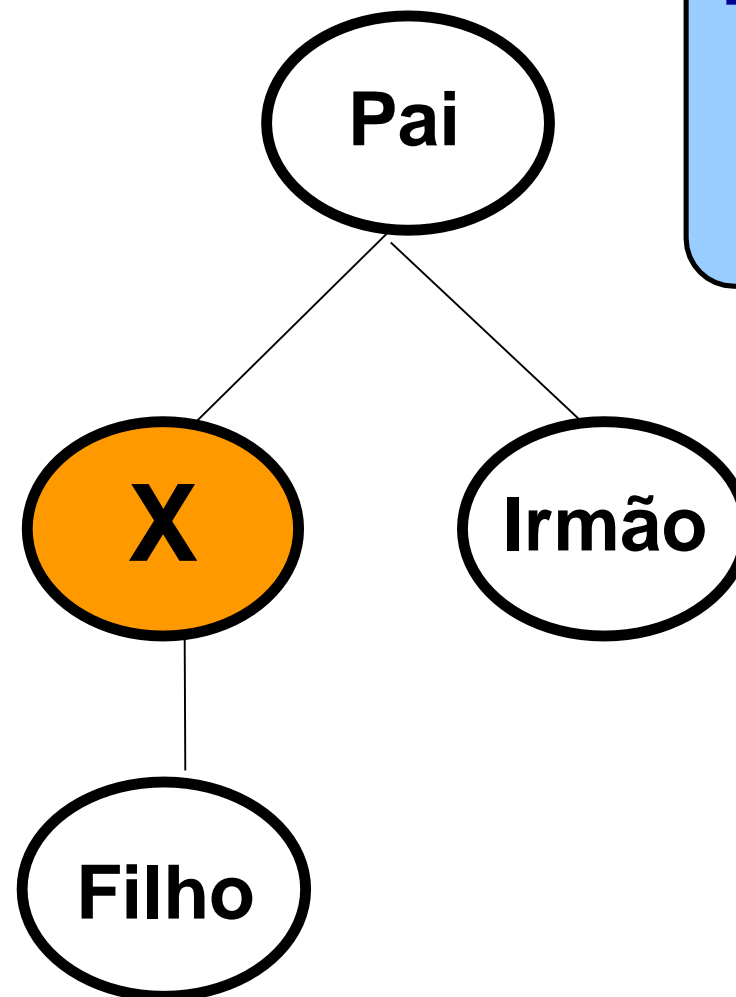
Raiz



Sub-árvores (de A)

TERMINOLOGIA

Usando o nó **X** como referencial



Pai

= mãe
= ascendente
= antecessor

Irmão

= irmã

Filho

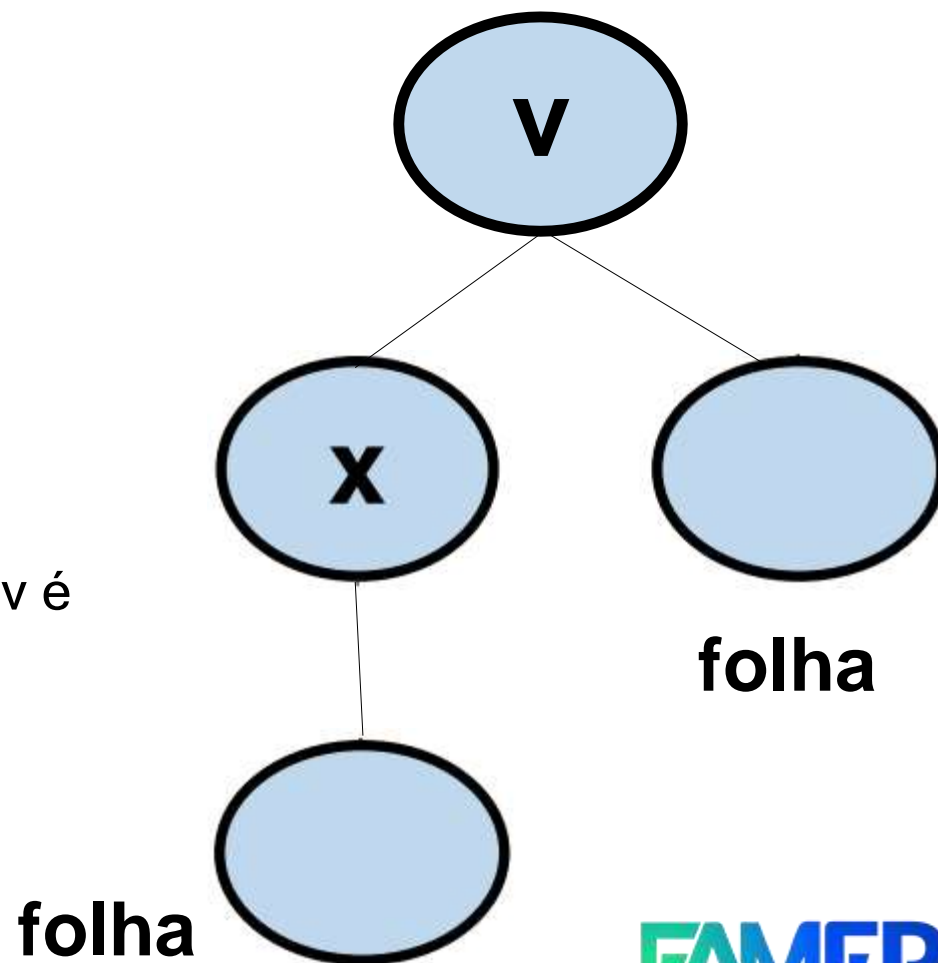
= filha
= descendente
= sucessor

TERMINOLOGIA

Um nó é ancestral e descendente de si mesmo

Se **x** pertence à subárvore enraizada em **v**:

- ☐ x é **descendente** de v
- ☐ v é **ancestral** de x
- ☐ Se x é diferente de v, x é **descendente próprio** de v, e v é **ancestral próprio** de x
- ☐ Um **nó folha** não possui descendentes próprios

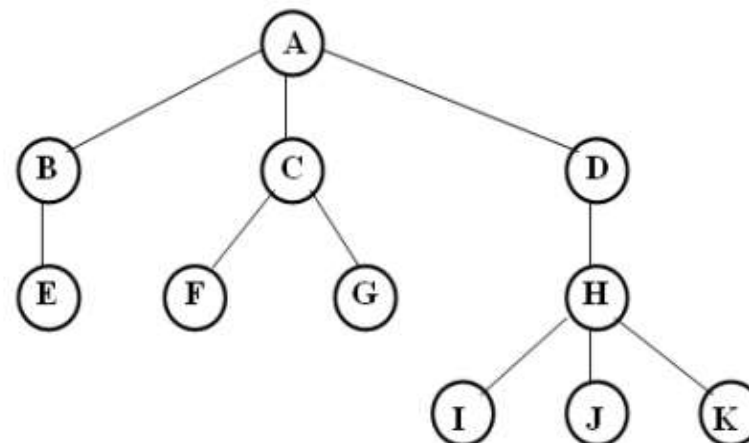


TERMINOLOGIA

Resumindo

- Um nó n_i é **ancestral** e **descendente** de si mesmo.
- Se há um caminho de n_i **para** n_j , então n_i é **ancestral de n_j** e n_j é **descendente de n_i** .
- Ancestral próprio ou descendente próprio de um nó é um ancestral ou descendente, respectivamente, diferente desse nó.

- Na árvore ao lado:
 - Ancestrais próprios de J: A, D, H
 - Descendentes próprios de D: H, I, J, K

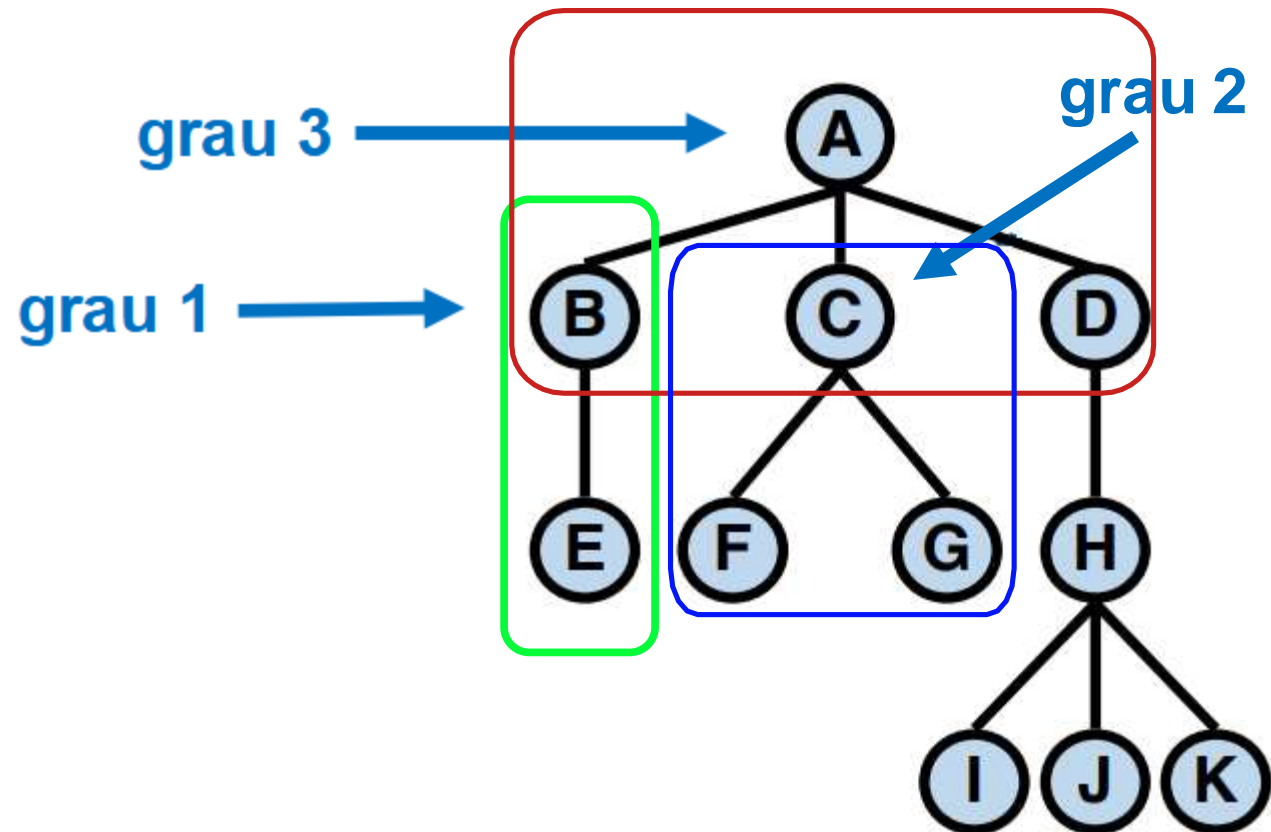


TERMINOLOGIA

Grau de um nó

Ou grau de saída

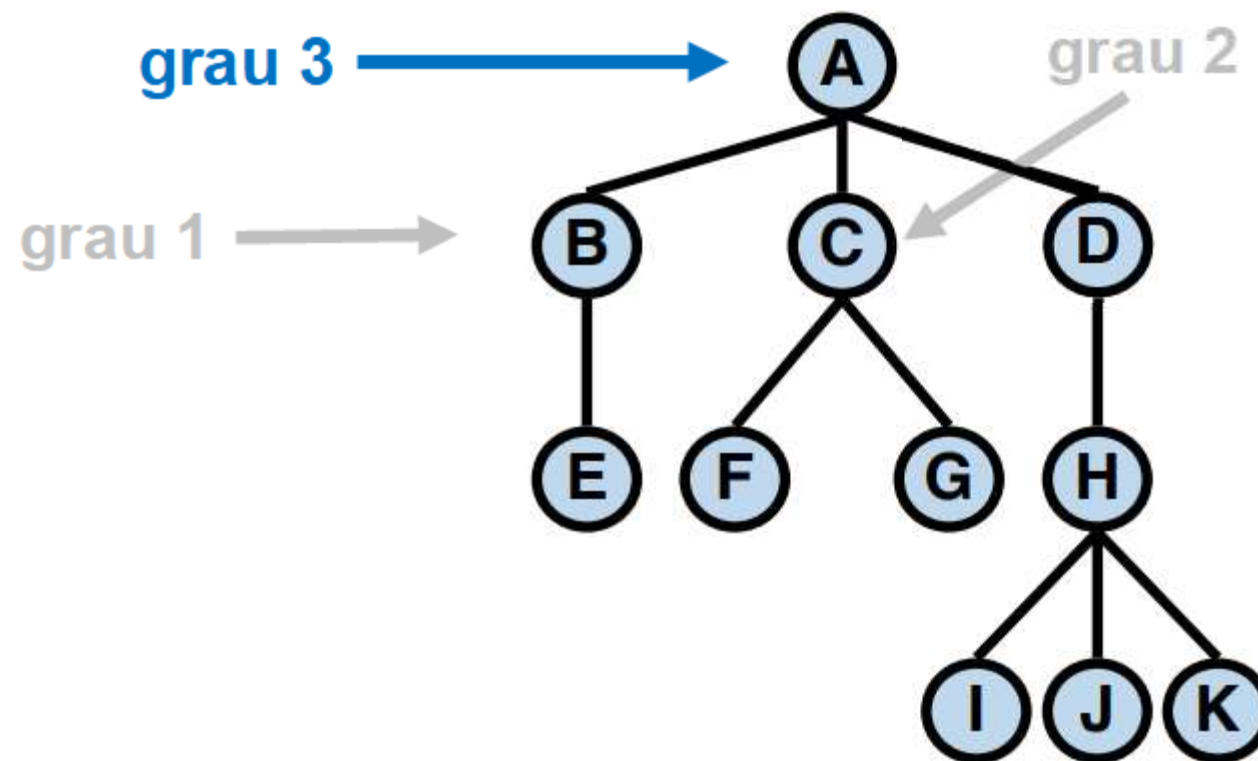
- Número de sub-árvores do nó ou
- Número de filhos de um nó



TERMINOLOGIA

Grau da árvore

Máximo entre os graus de seus nós

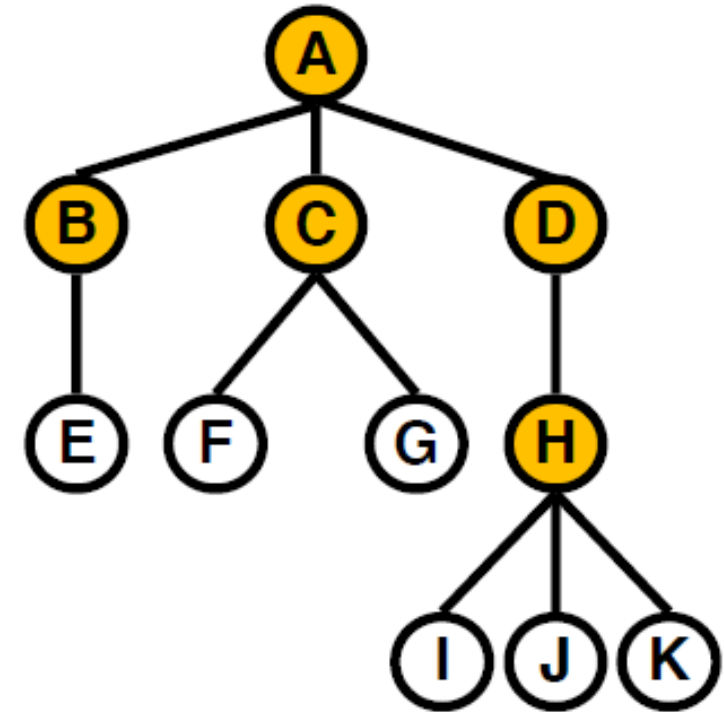


TERMINOLOGIA

Nó interno

Nó interno (ou nó de derivação)

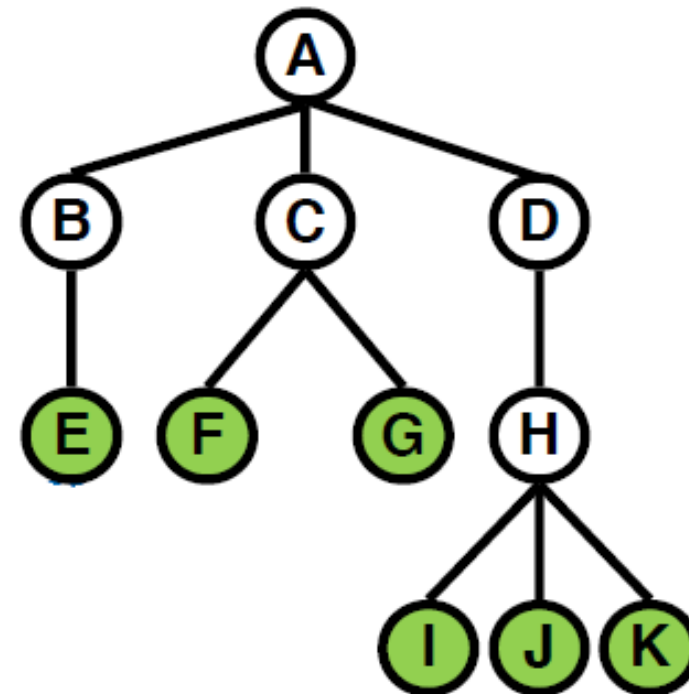
- nó com grau maior do que zero (tem pelo menos um filho)



TERMINOLOGIA

Nó folha

Nó folha (nó terminal ou externo)
nó com grau igual a zero (nó sem filhos)

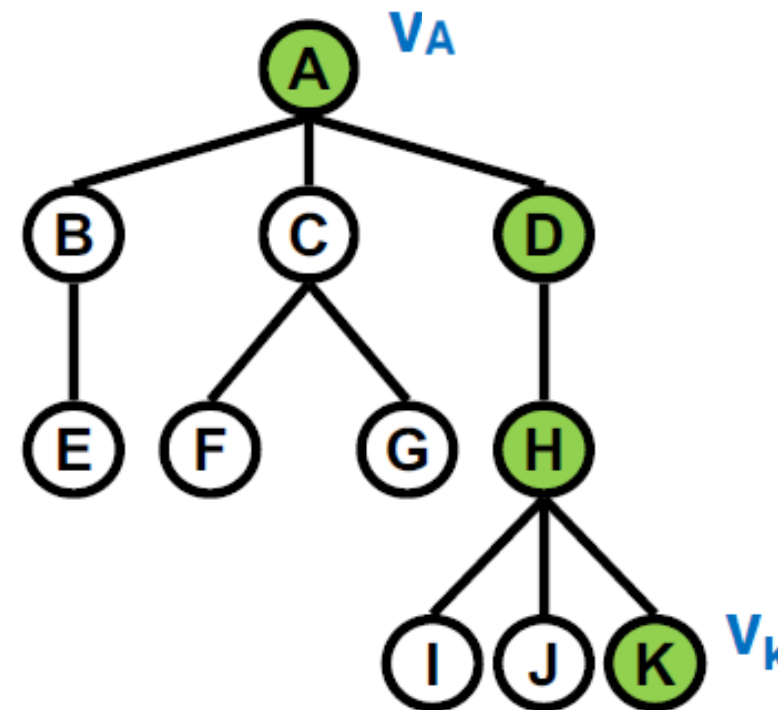


TERMINOLOGIA

Caminho

É uma sequência de nós consecutivos distintos entre dois nós

Caminho = A - D - H - K



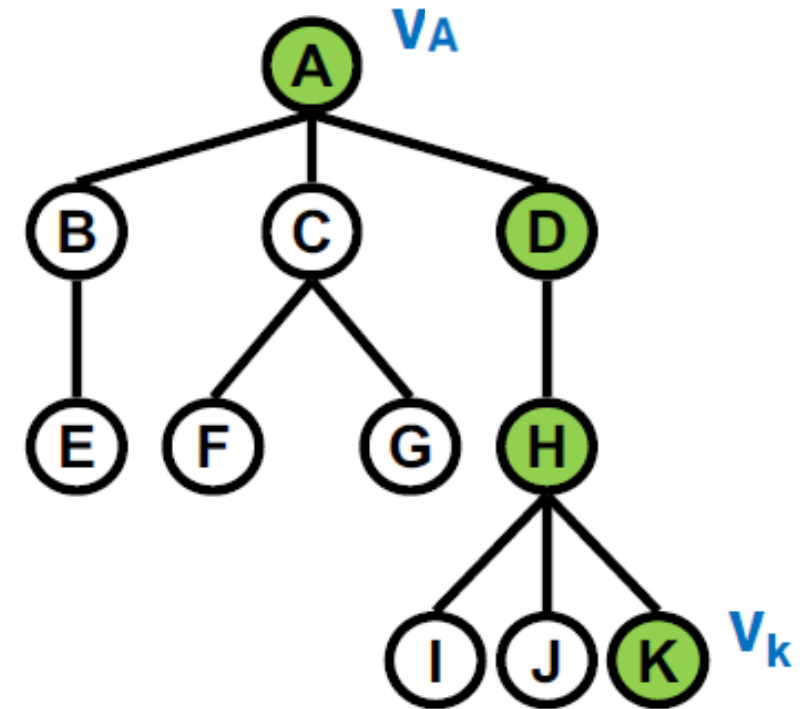
TERMINOLOGIA

Comprimento do caminho

Comprimento do caminho
é o número de ligações
(arestas) entre os
nós do caminho

É igual ao número de nós - 1

Comprimento do
caminho = 3

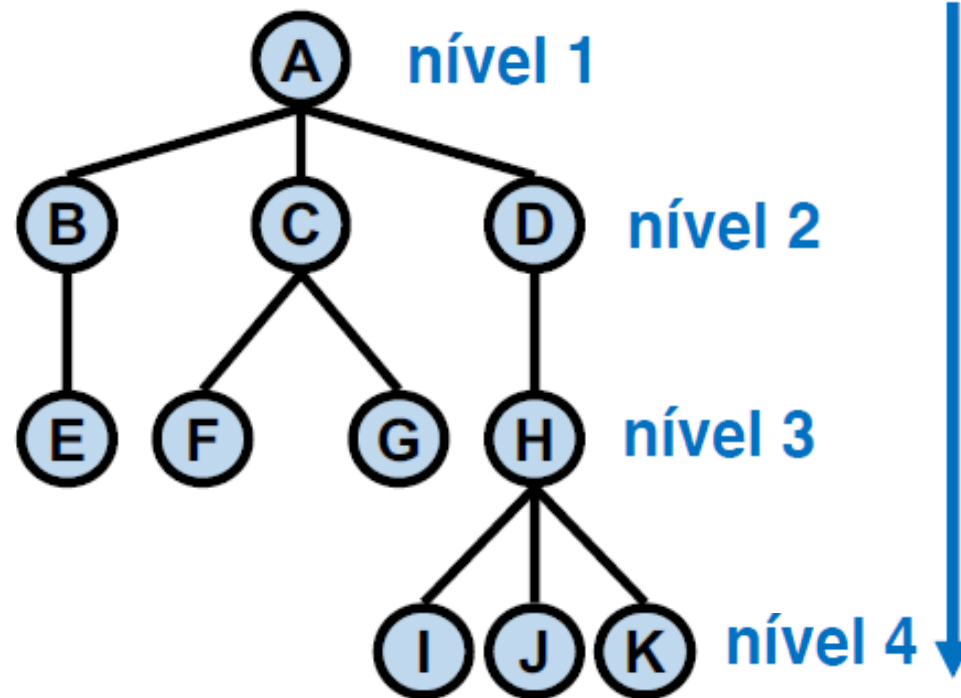


TERMINOLOGIA

Nível

Número de ligações entre a raiz e o nó, acrescido de uma unidade

É igual ao número de nós

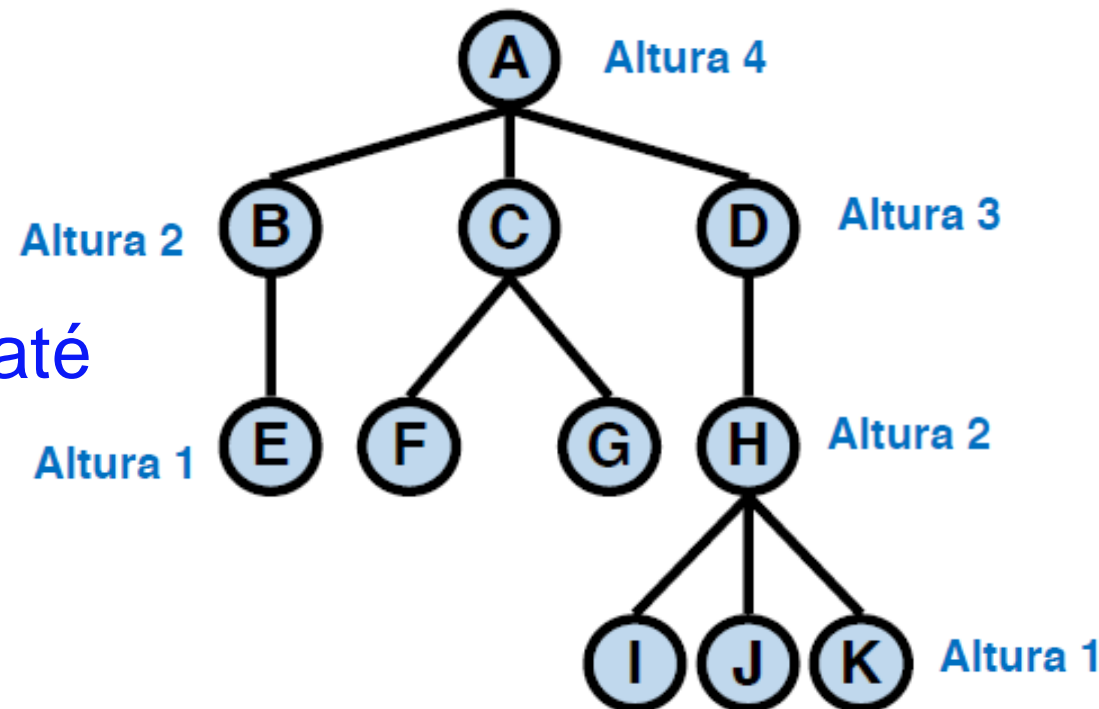


TERMINOLOGIA

Altura de um nó

Altura (profundidade) de um nó:
- número de ligações entre o nó até o nó folha descendente dele

Altura de um nó folha é igual a 1



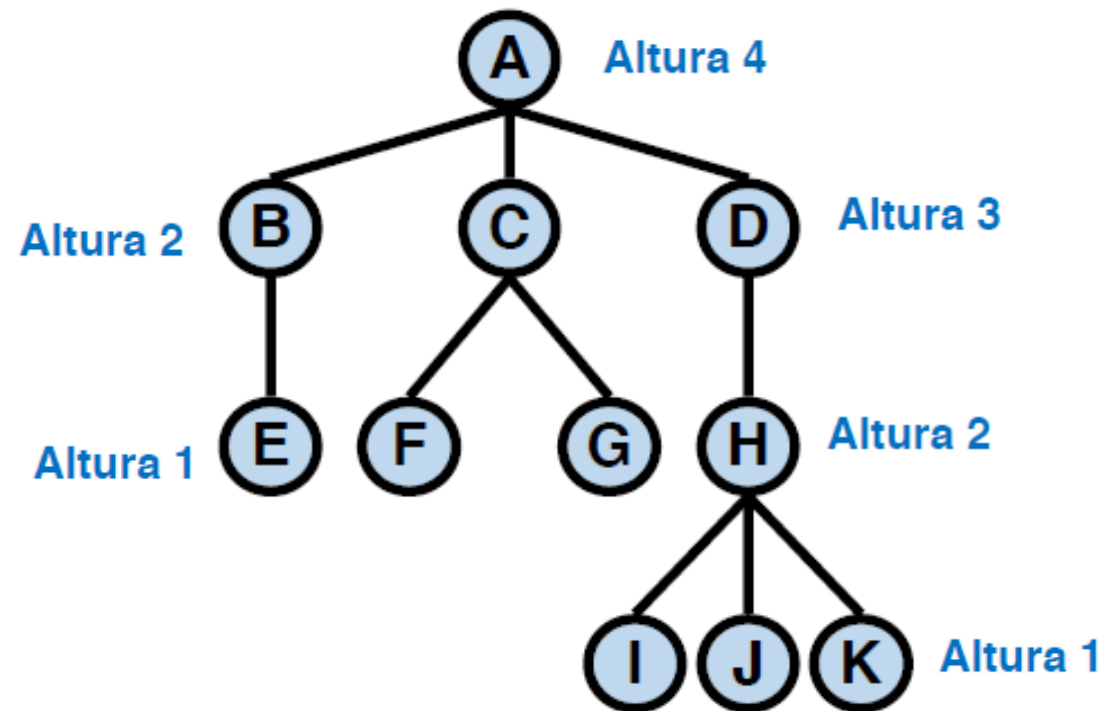
TERMINOLOGIA

Altura da árvore

Altura (profundidade) da árvore:

- Maior nível dentre seus nós

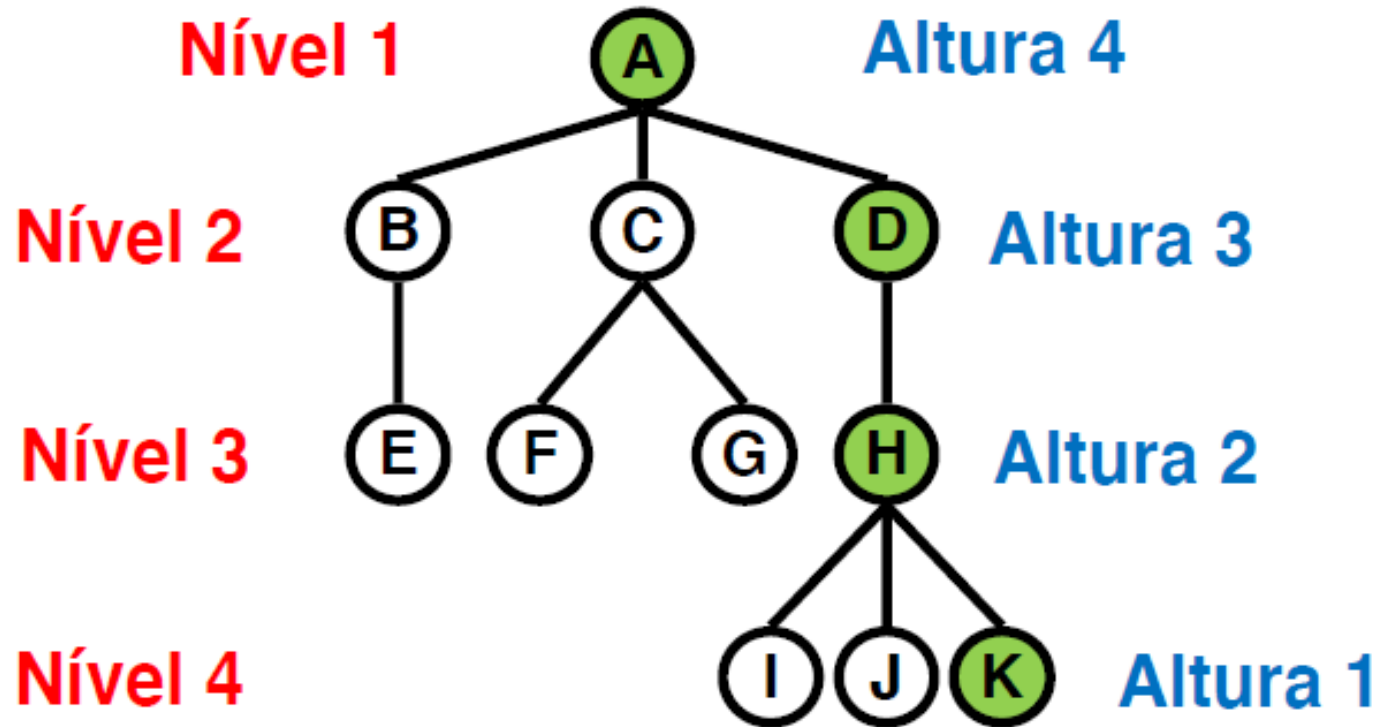
Equivale a altura do nó raiz



Altura da Árvore = 4

TERMINOLOGIA

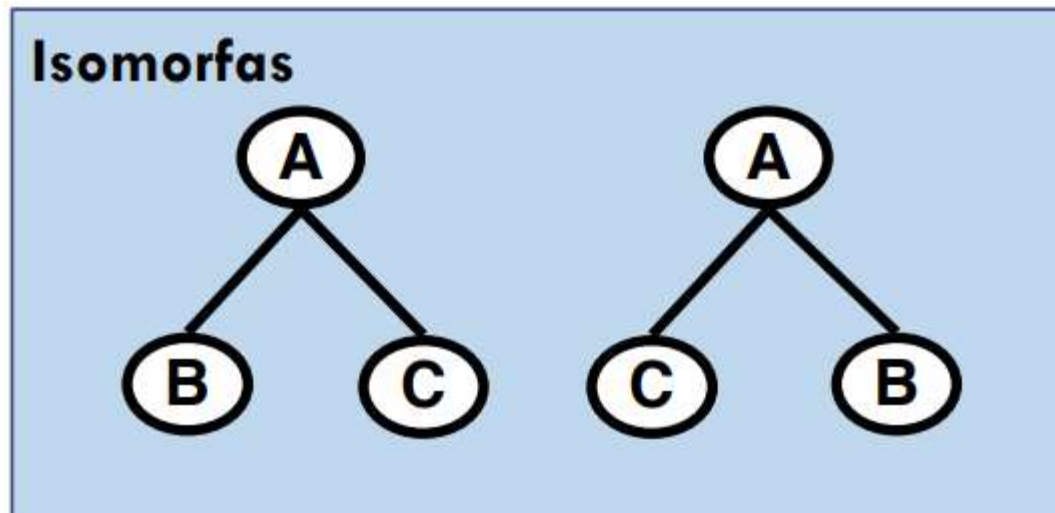
Altura X nível



TERMINOLOGIA

Árvore isomorfa

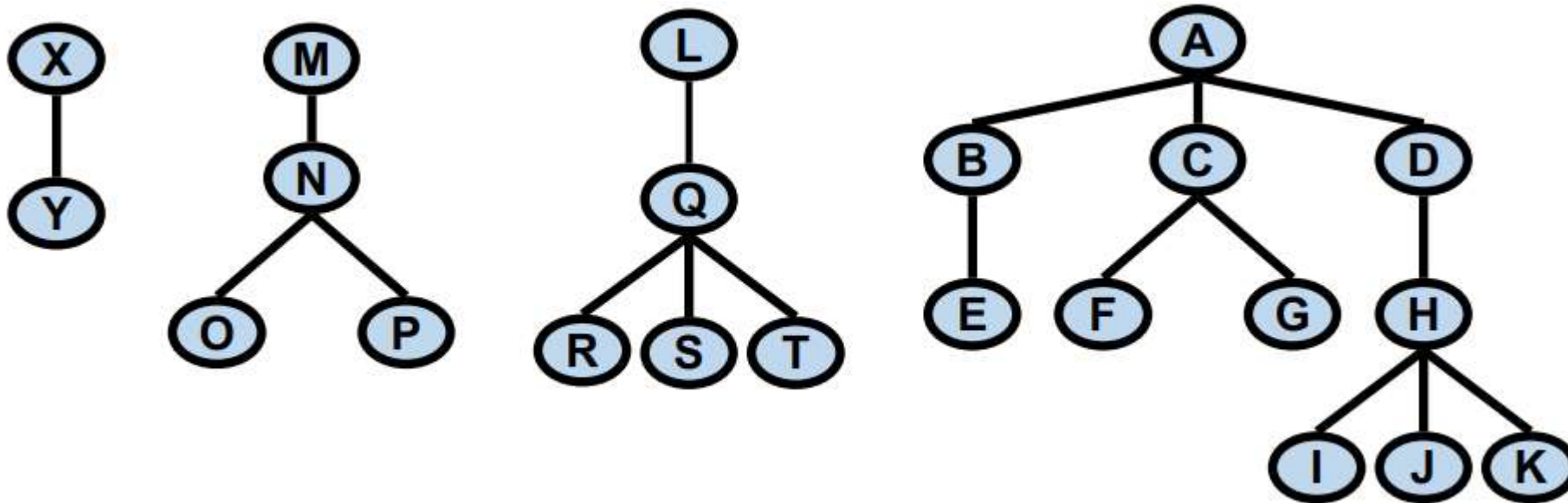
Duas árvores são **isomorfas** quando puderem se tornar **coincidentes** pela **permutação da ordem** das subárvores



TERMINOLOGIA

Floresta

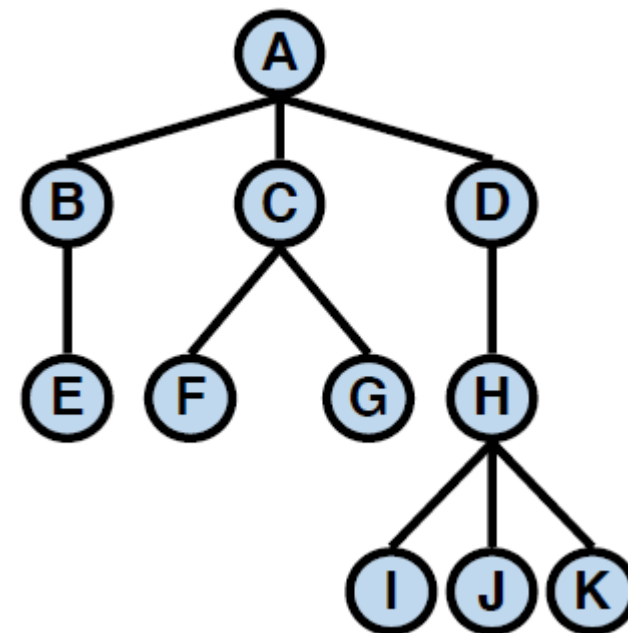
Floresta = Um conjunto de árvores



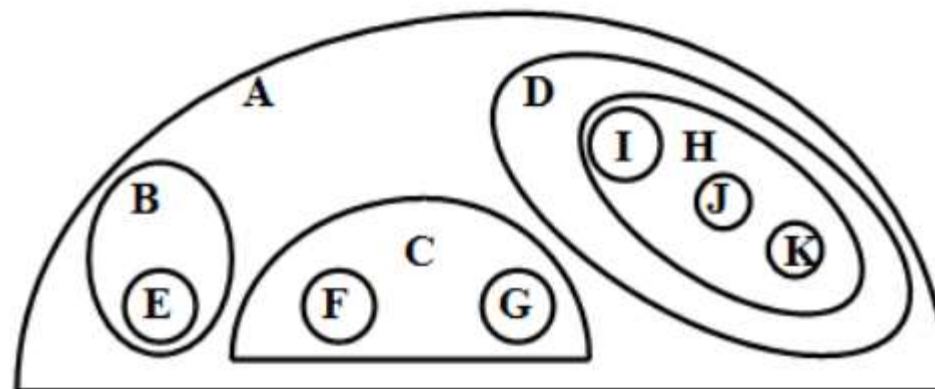
TERMINOLOGIA

Formas de representação

a) Forma convencional



b) Diagrama de conjuntos

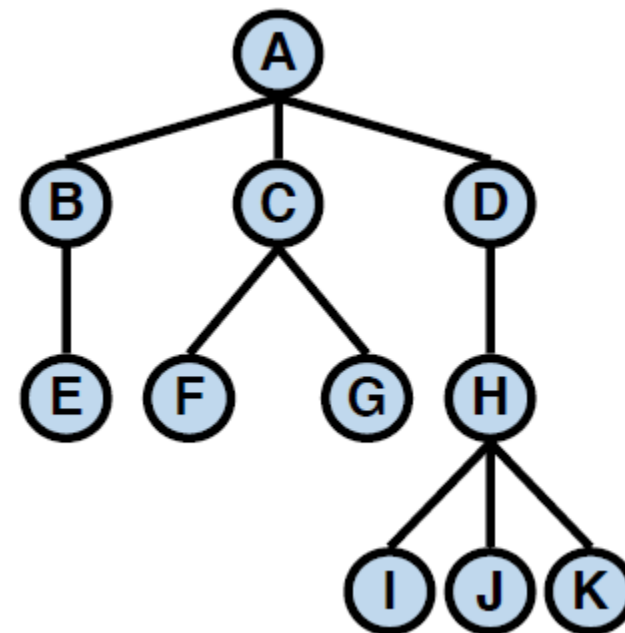


TERMINOLOGIA

Formas de representação

c) Forma parentética

Coloca-se entre parêntesis a raiz, seguida das formas parentéticas de suas sub-árvores, ordenadas da esquerda para a direita



(A (B (E)) (C (F) (G)) (D (H (I) (J) (K)))))

TERMINOLOGIA

Formas de representação

d) Forma tabulada

Diretoria

Departamento de Fabricação

Seção de Prensas

Seção de Tornos

Seção de Ferramentaria

Seção de Fornos

Seção de Banhos Químicos

Departamento de Engenharia de Produção

Seção de Desenvolvimento de Projetos

Seção de Desenhos

Seção de Controle de Qualidade

Departamento de Manutenção

Seção de Eletricistas

Seção de Mecânica

Seção de Hidráulica

Seção de Instalações Prediais

- Cada nó aparece numa linha, e seus filhos são listados com uma tabulação a mais que a desse nó.
 - Ex: organograma de uma empresa, índice de livros, etc.

TERMINOLOGIA

Formas de representação

e) Forma numerada (ou itemizada)

1 - Estruturas de dados

1.1 - Listas lineares

1.1.1 - Estrutura contígua

1.1.2 - Estrutura encadeada

1.1.3 - Pilhas e filas

1.2 - Árvores

1.2.1 - Definições

1.2.2 - Estruturas de dados

1.2.3 - Árvores binárias

1.3 - Grafos

1.3.1 - Grafos orientados

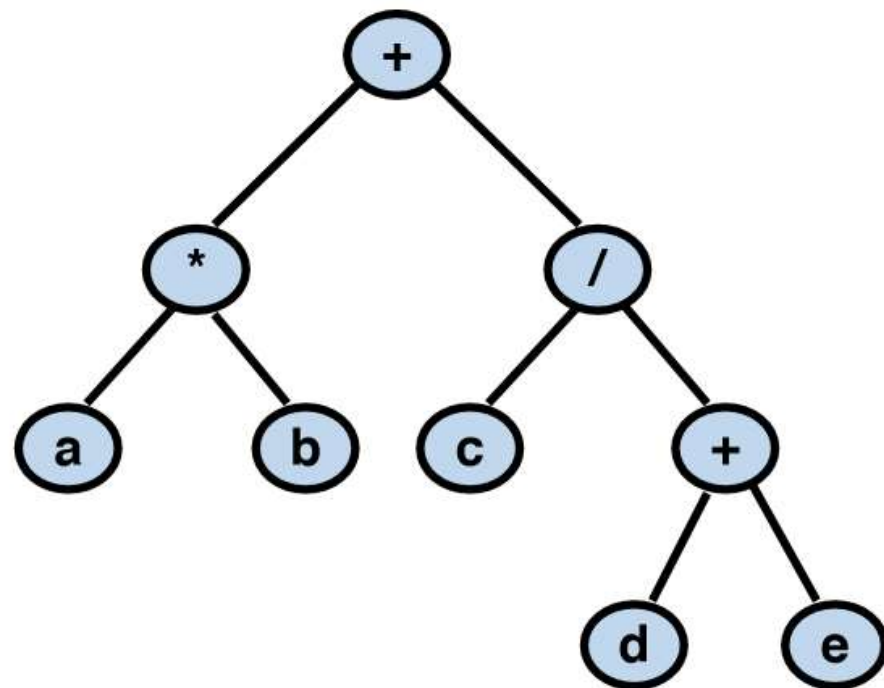
1.3.2 - Grafos não orientados

- Semelhante à anterior: a numeração de um nó tem como prefixo o número de seu pai, e como sufixo um número que o diferencie dos irmãos.

ÁRVORES BINÁRIAS

ÁRVORES BINÁRIAS

Árvores binárias são uma das árvores mais usadas em computação



Expressão aritmética: $(a * b) + (c / (d + e))$

ÁRVORES BINÁRIAS

DEFINIÇÃO

Conjunto finito T de zero ou mais nós (nodos), tal que:

Se número de nós é maior do que zero

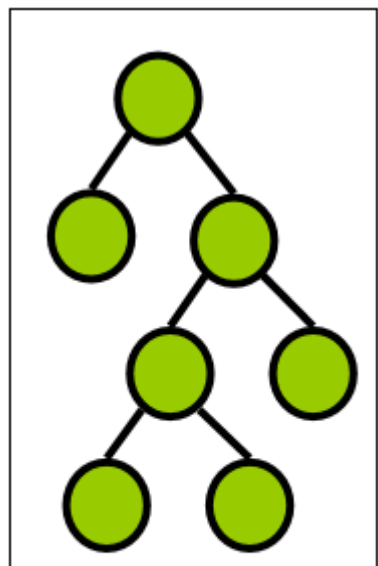
- existe um nó denominado **raiz** da árvore
- os demais nós formam 2 conjuntos disjuntos S_1 , S_2 (subárvore da esquerda e subárvore da direita) onde cada um destes é uma árvore binária

Se número de nós é igual a zero

- **árvore vazia**

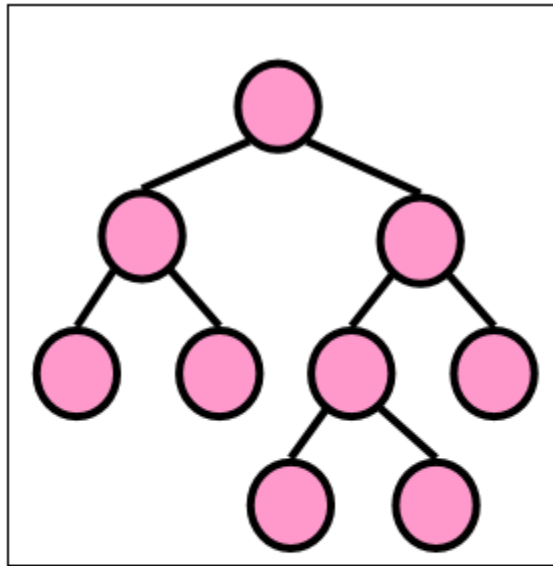
ÁRVORES BINÁRIAS

TIPOS DE ÁRVORES BINÁRIAS



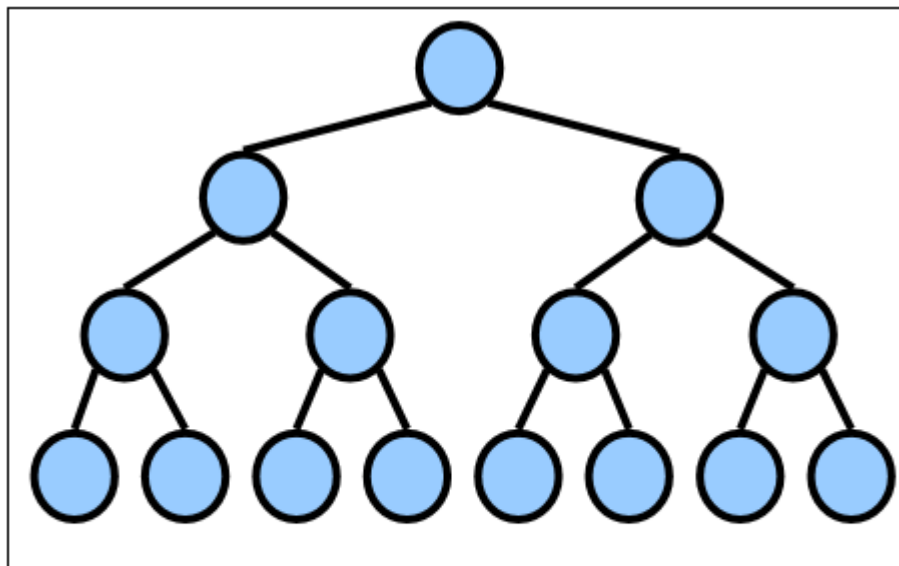
**Estritamente
Binária**

0 ou 2 filhos



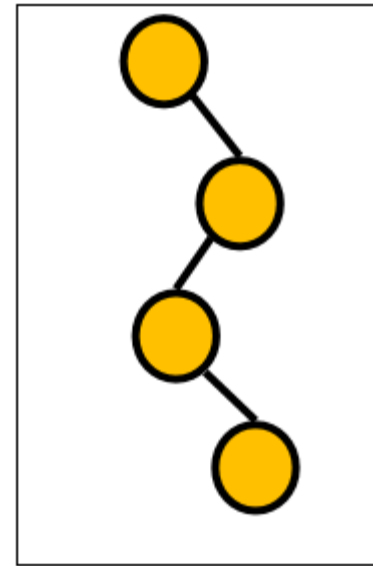
**Binária
Completa**

Sub-árvores vazias
apenas no último ou
penúltimo nível



**Binária
Cheia**

Sub-árvores vazias
somente no último nível



Zigue Zague

Nós internos com 1
subárvore vazia

ÁRVORES BINÁRIAS CAMINHAMENTOS

Um caminhoamento (ou percurso) define uma sequência de nós

Cada nó passa a ter um **nó seguinte**, ou um **nó anterior**, ou ambos (exceto árvore com 1 só nó)

Sequência de nós depende do caminhoamento

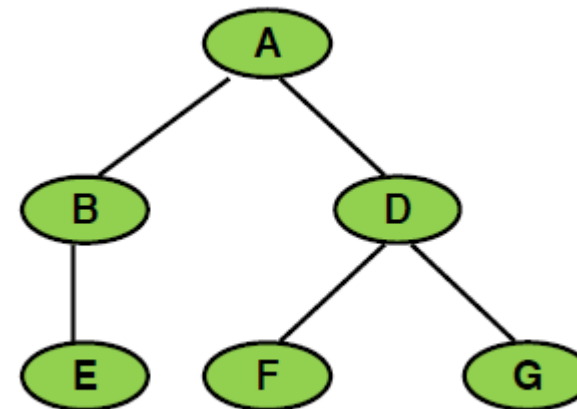
Exemplo:

Caminhamento 1:

- A – B – D – E – F – G

Caminhamento 2:

- A – B – E – D – F – G

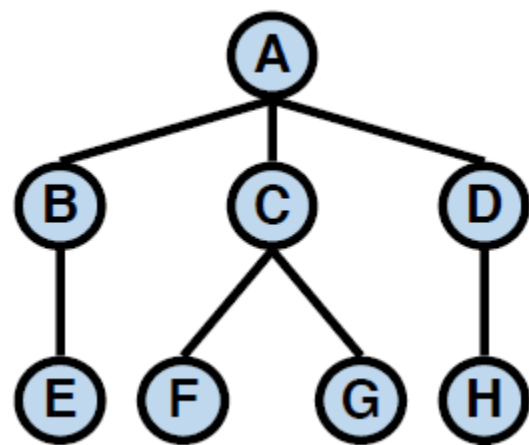


ÁRVORES BINÁRIAS CAMINHAMENTOS

- Ordenação de todos os nós de uma árvore
 - Existem formas de se ordenar e de se percorrer sistematicamente todos os nós de uma árvore:
 - Ordenação ou percurso por nível
 - Ordenação ou percurso em pré-ordem
 - Ordenação ou percurso em ordem central (ou in-ordem)
 - Ordenação ou percurso em pós-ordem

Busca em
largura

Busca em
profundidade



ÁRVORES BINÁRIAS CAMINHAMENTOS

A B C D E F G



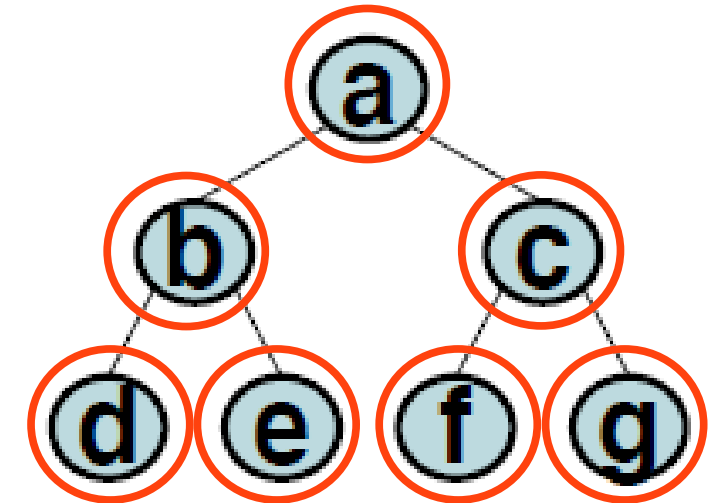
- Ordenação de todos os nós de uma árvore

- Ordenação ou percurso por nível

Busca em
largura

- Passos:

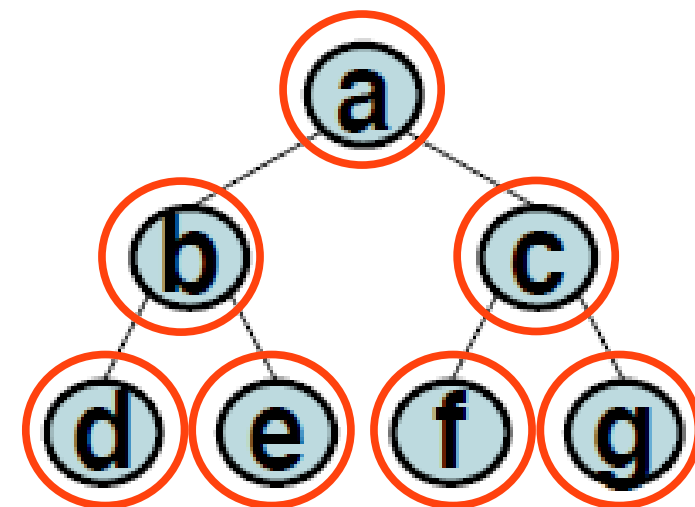
- Primeiramente, visita-se a raiz
- Depois visitam-se todos os filhos da raiz, da esquerda para a direita
- Depois os descendentes dos filhos da esquerda para a direita, e assim por diante.



ÁRVORES BINÁRIAS CAMINHAMENTOS

A B D E C F G

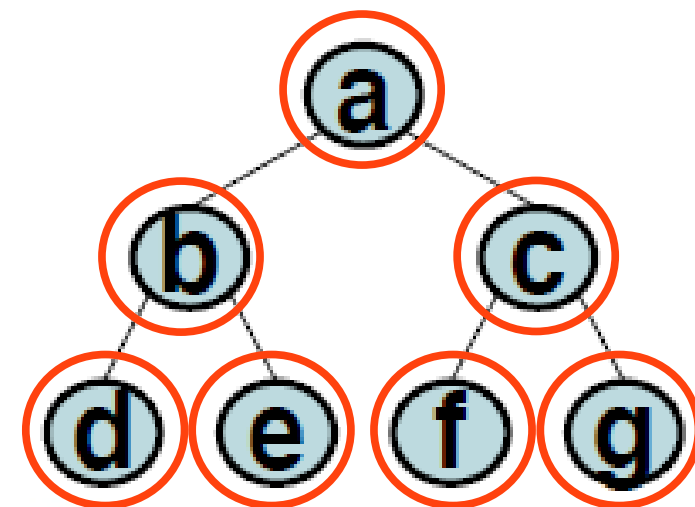
- Ordenação de todos os nós de uma árvore
 - Ordenação ou percurso em pré-ordem
 - Passos:
 - 1 Vistar a raiz.
 - 2 Percorrer a sua subárvore esquerda em pré-ordem.
 - 3 Percorrer a sua subárvore direita em pré-ordem.



ÁRVORES BINÁRIAS CAMINHAMENTOS

D B E A F C G

- Ordenação de todos os nós de uma árvore
 - Ordenação ou percurso em ordem central (ou in-ordem)
- Passos:
 - 1 Percorrer a sua subárvore esquerda em in-ordem.
 - 2 Vistar a raiz.
 - 3 Percorrer a sua subárvore direita em in-ordem.



É conhecida também pelo nome de ordem simétrica.

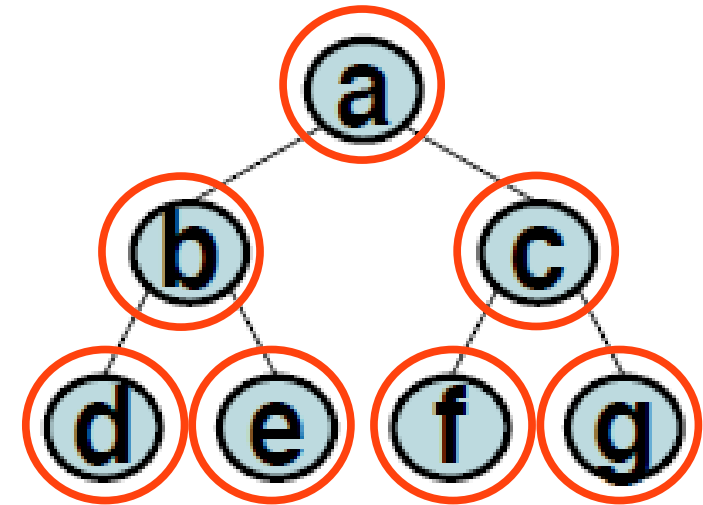
ÁRVORES BINÁRIAS CAMINHAMENTOS

D E B F G C A

- Ordenação de todos os nós de uma árvore
 - Ordenação ou percurso em pós-ordem

- Passos:

- 1 Percorrer a sua subárvore esquerda em pós-ordem.
- 2 Percorrer a sua subárvore direita em pós-ordem.
- 3 Vistar a raiz.





ÁRVORES BINÁRIAS

IMPLEMENTAÇÃO

ÁRVORES BINÁRIAS

- Definindo as estruturas

```
typedef struct tipo_item {  
    int cod;  
    char nome[30];  
    int quant;  
};
```

tipo_item

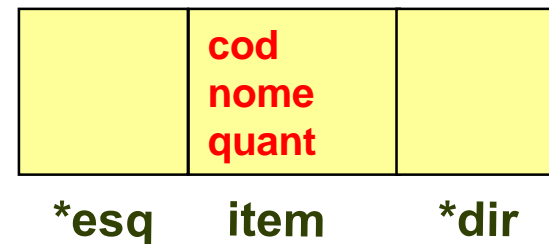
cod
nome
quant

ÁRVORES BINÁRIAS

- Definindo as estruturas

```
typedef struct tipo_no {  
    tipo_item item;  
    tipo_no *esq;  
    tipo_no *dir;  
};
```

tipo_no



ÁRVORES BINÁRIAS

Operações

inserir — Insere um item na árvore

mostrar — Mostra o conteúdo ordenado da árvore

pesquisar — Pesquisa um item na árvore

excluir — Remove um item da árvore

mostrarItem — mostra um item

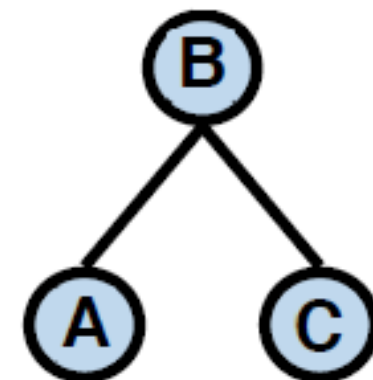
ÁRVORES BINÁRIAS

Regra da árvore binária de busca

Cada nó de referência tem palavras chaves (cod) nas seguintes condições:

= Menores à esquerda da raiz

= Maiores à direita da raiz




```

int main( ) {
    setlocale(LC_ALL, "");

    int op, cod, achou;

    tipo_no *raiz = NULL;

    tipo_item aux;

    do{
        system("cls");

        op = menu();

        switch(op){

            case 0: printf("\n\n\n\t Fim do Programa. \n\n");
                    getch();
                    return(0);

            case 1: aux = preencher_item();
                    if(!raiz) raiz = inserir(aux, raiz, raiz);
                    else inserir(aux, raiz, raiz);
                    break;

            case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");
                    mostrar(raiz);
                    getch(); // segura a tela
                    break;

            case 3: printf("\n\n PESQUISA NA ÁRVORE ");
                    printf("\n\n Digite o código do item a pesquisar: ");
                    scanf("%d", &cod);
                    achou = 0;
                    pesquisar(&achou, cod, raiz);
                    if(!achou) printf("\n\n O item de código %d não está na árvore", cod);
                    getch(); // segura a tela
                    break;
        }
    }while(1);
}

```

Código enviado com o slide

```

        case 4: printf("\n\n EXCLUIR DA ÁRVORE ");
                printf("\n\n Digite o código do item a excluir: ");
                scanf("%d", &cod);
                achou = 0;
                raiz = excluir(&achou, cod, raiz);
                if(!achou) printf("\n\n O item de código %d não está na árvore");
                else printf("\n\n O item de código %d foi excluído");
                getch(); // segura a tela
                break;

        default:
            printf("\n\n OPÇÃO INVÁLIDA !!!");
            getch();

        }// fim switch
    }while(1); //fim do-while
    return(0);
} // fim main

```

ÁRVORES BINÁRIAS

```
int main( ) {  
    setlocale(LC_ALL, "");
```

```
    int op, cod, achou;
```

```
    tipo_no *raiz = NULL;
```

```
    tipo_item aux;
```

```
    do{
```

```
        system("cls");
```

```
        op = menu();
```

```
        switch(op){
```

```
            case 0: printf("\n\n\n\t Fim do Programa. \n\n");  
                    getch();  
                    return(0);
```

```
            case 1: aux = preencher_item();  
                    if(!raiz) raiz = inserir(aux, raiz, raiz);  
                    else inserir(aux, raiz, raiz);  
                    break;
```

```
            case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                    mostrar(raiz);  
                    getch(); // segura a tela  
                    break;
```

```
            case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                    printf("\n\n Digite o código do item a pesquisar: ");  
                    scanf("%d", &cod);  
                    achou = 0;  
                    pesquisar(&achou, cod, raiz);  
                    if(!achou) printf("\n\n O item de código %d não está na árvore", cod);  
                    getch(); // segura a tela  
                    break;
```

op

cod

achou

ÁRVORES BINÁRIAS

```
int main() {  
    setlocale(LC_ALL, "");
```

op

cod

achou

```
    int op, cod, achou;
```

```
    tipo_no *raiz = NULL;
```

*rai

NULL

```
    tipo_item aux;
```

```
do{
```

```
    system("cls");
```

```
    op = menu();
```

```
    switch(op){
```

```
        case 0: printf("\n\n\n\t Fim do Programa. \n\n");  
                getch();  
                return(0);
```

```
        case 1: aux = preencher_item();  
                if(!raiz) raiz = inserir(aux, raiz, raiz);  
                else inserir(aux, raiz, raiz);  
                break;
```

```
        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                mostrar(raiz);  
                getch(); // segura a tela  
                break;
```

```
        case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                printf("\n\n Digite o código do item a pesquisar: ");  
                scanf("%d", &cod);  
                achou = 0;  
                pesquisar(&achou, cod, raiz);  
                if(!achou) printf("\n\n O item de código %d não está na árvore", cod);  
                getch(); // segura a tela  
                break;
```

ÁRVORES BINÁRIAS

```
int main() {  
    setlocale(LC_ALL, "");
```

op

cod

achou

```
    int op, cod, achou;
```

```
    tipo_no *raiz = NULL;
```

```
    tipo_item aux;
```

*rai

NULL

aux

cod
nome
quant

```
do{
```

```
    system("cls");
```

```
    op = menu();
```

```
    switch(op){
```

```
        case 0: printf("\n\n\n\t Fim do Programa. \n\n");  
                getch();  
                return(0);
```

```
        case 1: aux = preencher_item();  
                if(!raiz) raiz = inserir(aux, raiz, raiz);  
                else inserir(aux, raiz, raiz);  
                break;
```

```
        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                mostrar(raiz);  
                getch(); // segura a tela  
                break;
```

```
        case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                printf("\n\n Digite o código do item a pesquisar: ");  
                scanf("%d", &cod);  
                achou = 0;  
                pesquisar(&achou, cod, raiz);  
                if(!achou) printf("\n\n O item de código %d não está na árvore", cod);  
                getch(); // segura a tela  
                break;
```



ÁRVORES BINÁRIAS

IMPLEMENTAÇÃO

INSERIR NA ÁRVORE

ÁRVORES BINÁRIAS

Procedimento para Inserir na Árvore

- Onde inserir?
 - Atingir um ponteiro nulo em um processo de pesquisa significa uma pesquisa sem sucesso.
 - O ponteiro nulo atingido é o ponto de inserção.
- Como inserir?
 1. Cria registro.
 2. Procurar o lugar na árvore.
 3. Se registro não tiver na árvore, insere-o.

ÁRVORES BINÁRIAS

Procedimento para inserir na árvore

- Onde inserir?
 - Atingir um ponteiro nulo em um processo de pesquisa significa uma pesquisa sem sucesso.
 - O ponteiro nulo atingido é o ponto de inserção.
- Como inserir?
 1. Cria registro.
 2. Procurar o lugar na árvore.
 3. Se registro não tiver na árvore, insere-o.

ÁRVORES BINÁRIAS

```
int main() {
    setlocale(LC_ALL, "");

    int op, cod, achou;

    tipo_no *raiz = NULL;

    tipo_item aux;
```

op

cod

achou

*rai

NULL

aux

cod
nome
quant

```
int menu(){
    int op;

    printf("\n\n ***** MENU *****");

    printf("\n [1] - INSERIR");
    printf("\n [2] - MOSTRAR");
    printf("\n [3] - PESQUISAR");
    printf("\n [4] - EXCLUIR");

    printf("\n [0] - SAIR");

    printf("\n\n Digite sua opção: ");
    scanf("%d", &op);

    return(op);
}
```

```
do{
    system("cls");

    op = menu();

    switch(op){

        case 0: printf("\n\n\n\t Fim do Programa. \n\n");
                getch();
                return(0);

        case 1: aux = preencher_item();
                if(!raiz) raiz = inserir(aux, raiz, raiz);
                else inserir(aux, raiz, raiz);
                break;

        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");
                mostrar(raiz);
                getch(); // segura a tela
                break;

        case 3: printf("\n\n PESQUISA NA ÁRVORE ");
                printf("\n\n Digite o código do item a pesquisar: ");
                scanf("%d", &cod);
                achou = 0;
                pesquisar(&achou, cod, raiz);
                if(!achou) printf("\n\n O item de código %d não está na árvore", c
                getch(); // segura a tela
                break;
```

```
***** MENU *****
[1] - INSERIR
[2] - MOSTRAR
[3] - PESQUISAR
[4] - EXCLUIR
[0] - SAIR

Digite sua opção: 1
```


ÁRVORES BINÁRIAS

```
tipo_item preencher_item(){
    tipo_item item;
    printf("\n\n ***** PREENCHENDO ITEM A SER INSERIDO *****");

    printf("\n\n Digite o código: ");
    scanf("%d", &item.cod);

    fflush(stdin); // limpar o buffer do teclado
    printf("\n\n Digite o nome do item: ");
    gets(item.nome);

    printf("\n\n Digite a quantidade: ");
    scanf("%d", &item.quant);

    return(item);
}
```

```
int main() {
    setlocale(LC_ALL, "");

    int op, cod, achou;

    tipo_no *raiz = NULL;

    tipo_item aux;
```

```
do{
    system("cls");

    op = menu();

    switch(op){

        case 0: printf("\n\n\n\t Fim do Programa. \n\n");
                getch();
                return(0);

        case 1: aux = preencher_item();
                if(!raiz) raiz = inserir(aux, raiz, raiz);
                else inserir(aux, raiz, raiz);
                break;

        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");
                mostrar(raiz);
                getch(); // segura a tela
                break;

        case 3: printf("\n\n PESQUISA NA ÁRVORE ");
                printf("\n\n Digite o código do item a pesquisar: ");
                scanf("%d", &cod);
                achou = 0;
                pesquisar(&achou, cod, raiz);
                if(!achou) printf("\n\n O item de código %d não está na árvore", cod);
                getch(); // segura a tela
                break;
```

op

1

cod

achou

*rai

NULL

aux

cod
nome
quant

```

***** PREENCHENDO ITEM A SER INSERIDO *****

Digite o código: 5

Digite o nome do item: Arroz

Digite a quantidade: 500

```

Item

cod	nome	quant
-----	------	-------

```

tipo_item preencher_item(){

    tipo_item item;
    printf("\n\n ***** PREENCHENDO ITEM A SER INSERIDO *****");

    printf("\n\n Digite o código: ");
    scanf("%d", &item.cod);

    fflush(stdin); // limpar o buffer do teclado
    printf("\n\n Digite o nome do item: ");
    gets(item.nome);

    printf("\n\n Digite a quantidade: ");
    scanf("%d", &item.quant);

    return(item);
}

```

```

int main() {
    setlocale(LC_ALL, "");

    int op, cod, achou;

    tipo_no *raiz = NULL;

    tipo_item aux;

```

op

1

cod

achou

*raiz

NULL

aux

cod	nome	quant
-----	------	-------

```

do{
    system("cls");

    op = menu();

    switch(op){

        case 0: printf("\n\n\n\t Fim do Programa. \n\n");
                getch();
                return(0);

        case 1: aux = preencher_item();
                if(!raiz) raiz = inserir(aux, raiz, raiz);
                else inserir(aux, raiz, raiz);
                break;

        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");
                mostrar(raiz);
                getch(); // segura a tela
                break;

        case 3: printf("\n\n PESQUISA NA ÁRVORE ");
                printf("\n\n Digite o código do item a pesquisar: ");
                scanf("%d", &cod);
                achou = 0;
                pesquisar(&achou, cod, raiz);
                if(!achou) printf("\n\n O item de código %d não está na árvore", cod);
                getch(); // segura a tela
                break;
    }
}

```

ÁRVORES BINÁRIAS

Item 5
Arroz
500

```
tipo_item preencher_item(){  
    tipo_item item;  
    printf("\n\n ***** PREENCHENDO ITEM A SER INSERIDO *****");  
  
    printf("\n\n Digite o código: ");  
    scanf("%d", &item.cod);  
  
    fflush(stdin); // limpar o buffer do teclado  
    printf("\n\n Digite o nome do item: ");  
    gets(item.nome);  
  
    printf("\n\n Digite a quantidade: ");  
    scanf("%d", &item.quant);  
  
    return(item);  
}
```

```
int main() {  
    setlocale(LC_ALL, "");
```

```
    int op, cod, achou;
```

```
    tipo_no *raiz = NULL;
```

```
    tipo_item aux;
```

```
    do{
```

```
        system("cls");
```

```
        op = menu();
```

```
        switch(op){
```

```
            case 0: printf("\n\n\n\t Fim do Programa. \n\n");  
                    getch();  
                    return(0);
```

```
            case 1: aux = preencher_item();  
                    if(!raiz) raiz = inserir(aux, raiz, raiz);  
                    else inserir(aux, raiz, raiz);  
                    break;
```

```
            case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                    mostrar(raiz);  
                    getch(); // segura a tela  
                    break;
```

```
            case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                    printf("\n\n Digite o código do item a pesquisar: ");  
                    scanf("%d", &cod);  
                    achou = 0;  
                    pesquisar(&achou, cod, raiz);  
                    if(!achou) printf("\n\n O item de código %d não está na árvore", cod);  
                    getch(); // segura a tela  
                    break;
```

op

1

cod

achou

*raiz

NULL

aux

cod
nome
quant

ÁRVORES BINÁRIAS

Item **5**
Arroz
500

```
tipo_item preencher_item(){
    tipo_item item;
    printf("\n\n ***** PREENCHENDO ITEM A SER INSERIDO *****");

    printf("\n\n Digite o código: ");
    scanf("%d", &item.cod);

    fflush(stdin); // limpar o buffer do teclado
    printf("\n\n Digite o nome do item: ");
    gets(item.nome);

    printf("\n\n Digite a quantidade: ");
    scanf("%d", &item.quant);

    return(item);
}
```

```
int main() {
    setlocale(LC_ALL, "");

    int op, cod, achou;

    tipo_no *raiz = NULL;

    tipo_item aux;
```

```
do{
    system("cls");
```

```
    op = menu();
```

```
    switch(op){
```

```
        case 0: printf("\n\n\n\t Fim do Programa. \n\n");
                 getch();
                 return(0);
```

```
        case 1: aux = preencher_item();
                 if(!raiz) raiz = inserir(aux, raiz, raiz);
                 else inserir(aux, raiz, raiz);
                 break;
```

```
        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");
                 mostrar(raiz);
                 getch(); // segura a tela
                 break;
```

```
        case 3: printf("\n\n PESQUISA NA ÁRVORE ");
                 printf("\n\n Digite o código do item a pesquisar: ");
                 scanf("%d", &cod);
                 achou = 0;
                 pesquisar(&achou, cod, raiz);
                 if(!achou) printf("\n\n O item de código %d não está na árvore", cod);
                 getch(); // segura a tela
                 break;
```

op

1

cod

achou

*rai

NULL

aux

5
Arroz
500

ÁRVORES BINÁRIAS

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }

        r->esq = NULL;
        r->dir = NULL;
        //r->esq = r->dir = NULL; // atribuição múltipla
        r->item = x;

        if(!raiz){
            printf("\n\n Item inserido com sucesso!!!");
            getch();
            return r; // primeira entrada => raiz da árvore
        }

        if(x.cod < raiz->item.cod) raiz->esq = r;

        else raiz->dir = r;

        printf("\n\n Item inserido com sucesso!!!");
        getch();

        return r;
    }

    if(x.cod < r->item.cod) inserir(x, r, r->esq);

    else if(x.cod > r->item.cod) inserir(x, r, r->dir);

    else{
        printf("\n\n IMPOSSÍVEL INSERIR !!!");
        printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
        getch(); // segura a tela
    }
} // fim inserir()
```

```
int main() {
    setlocale(LC_ALL, "");

    int op, cod, achou;

    tipo_no *raiz = NULL;

    tipo_item aux;
```

```
do{
    system("cls");

    op = menu();
```

```
switch(op){
```

```
    case 0: printf("\n\n\n\t Fim do Programa. \n\n");
             getch();
             return(0);
```

```
    case 1: aux = preencher_item();
             if(!raiz) raiz = inserir(aux, raiz, raiz);
             else inserir(aux, raiz, raiz);
             break;
```

```
    case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");
             mostrar(raiz);
             getch(); // segura a tela
             break;
```

```
    case 3: printf("\n\n PESQUISA NA ÁRVORE ");
             printf("\n\n Digite o código do item a pesquisar: ");
             scanf("%d", &cod);
             achou = 0;
             pesquisar(&achou, cod, raiz);
             if(!achou) printf("\n\n O item de código %d não está na árvore", cod);
             getch(); // segura a tela
             break;
```

op

1

cod

achou

*rai

NULL

aux

5
Arroz
500

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

NULL

x

5
Arroz
500

op

1

cod

*rai

NULL

achou

aux

5
Arroz
500

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }
    }
```

```
    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;
```

```
    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");
    getch();
```

```
    return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

NULL

X

5
Arroz
500

op

1

cod

achou

*rai

NULL

aux

5
Arroz
500

*esq

item

*dir

55A2C4

*esq	item	*dir
	cod nome quant	

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }
    }
```

```
    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;
```

```
    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");
    getch();
```

```
    return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

55A2C4

X

5

Arroz
500

op

1

cod

achou

*rai

NULL

aux

5
Arroz
500

55A2C4

*esq	item	*dir
	cod nome quant	

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```



```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }
    }
```

```
    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;
```

```
    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");
    getch();
```

```
    return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x , r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x , r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

55A2C4

X

5

Arroz
500

op

1

cod

achou

*rai

NULL

aux

5
Arroz
500

55A2C4

*esq	item	*dir
	cod nome quant	

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

55A2C4

5

Arroz
500

X

op

1

cod

*rai

NULL

achou

aux

5
Arroz
500

55A2C4

*esq	item	*dir
NUL	5 Arroz 500	NUL

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }
    }
```

```
    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;
```

```
    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");
    getch();
```

```
    return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

55A2C4

X

5

Arroz
500

op

1

cod

achou

*rai

NULL

aux

5
Arroz
500

55A2C4

*esq	item	*dir
NUL L	5 Arroz 500	NUL L

Item inserido com sucesso!!!

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }
    }
```

```
    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;
```

```
    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");
    getch();
```

```
    return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```

*rai

NULL

*

55A2C4

X

5

Arroz
500

op

1

cod

achou

*rai

55A2C4

aux

5
Arroz
500

55A2C4

*esq

item

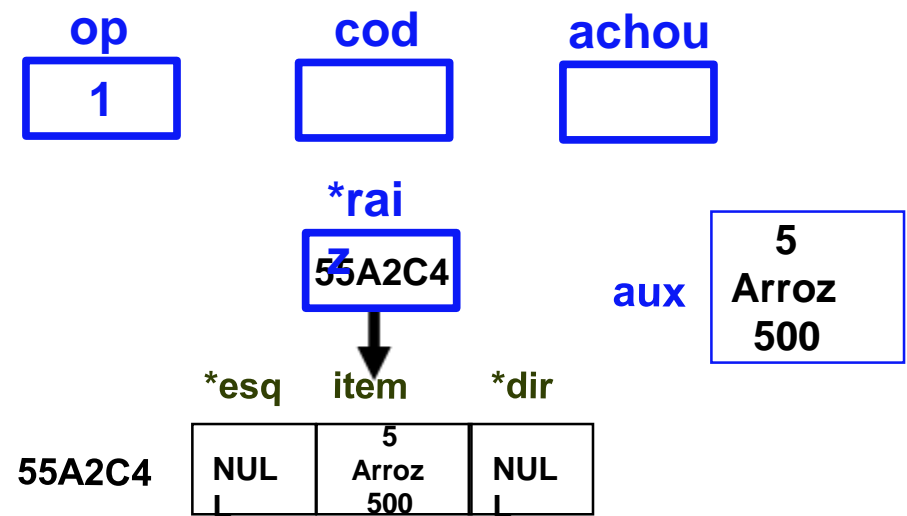
*dir

NUL	5 Arroz 500	NUL
L		L

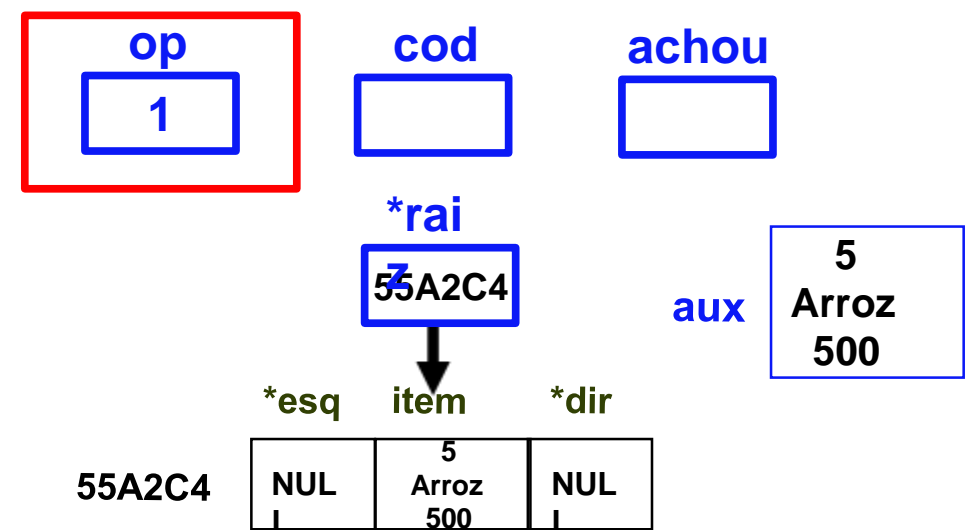
Item inserido com sucesso!!!

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

ÁRVORES BINÁRIAS



ÁRVORES BINÁRIAS

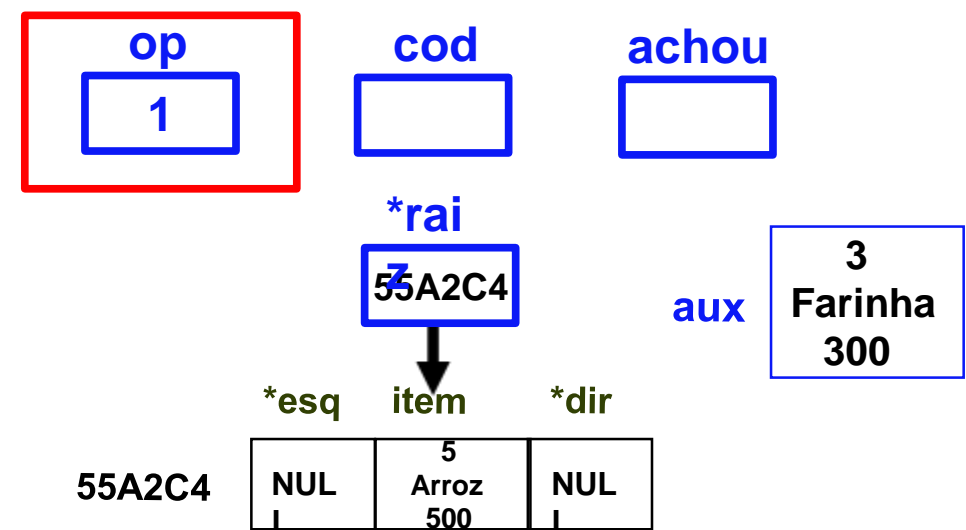


```
***** MENU *****
[1] - INSERIR
[2] - MOSTRAR
[3] - PESQUISAR
[4] - EXCLUIR
[0] - SAIR

Digite sua opção: 1
```

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

ÁRVORES BINÁRIAS



```
***** MENU *****
[1] - INSERIR
[2] - MOSTRAR
[3] - PESQUISAR
[4] - EXCLUIR
[0] - SAIR

Digite sua opção: 1
```

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }

    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;

    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }

    if(x.cod < raiz->item.cod) raiz->esq = r;

    else raiz->dir = r;

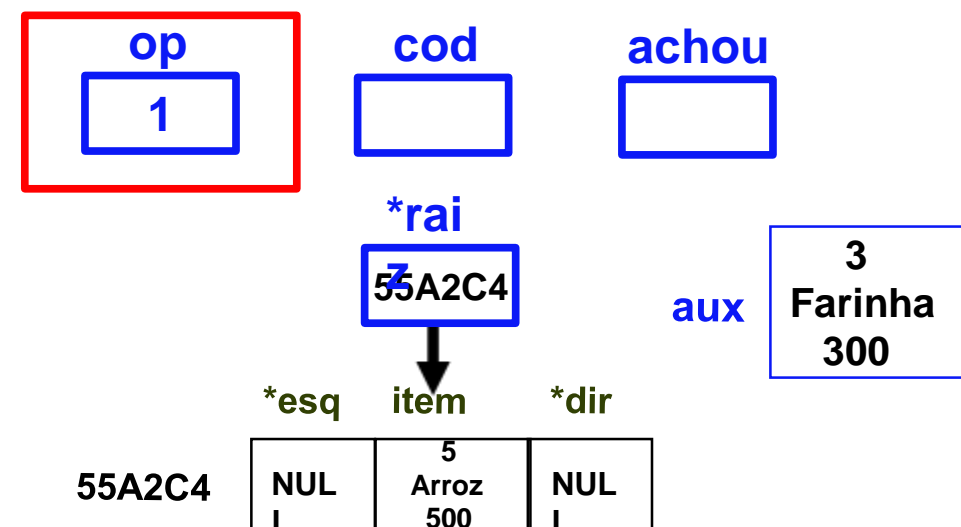
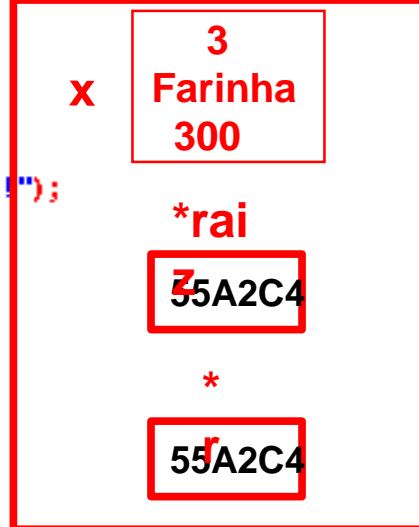
    printf("\n\n Item inserido com sucesso!!!");
    getch();

    return r;
}

if(x.cod < r->item.cod) inserir(x , r, r->esq);

else if(x.cod > r->item.cod) inserir(x , r, r->dir);

else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
} // fim inserir()
```



```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```



```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
```

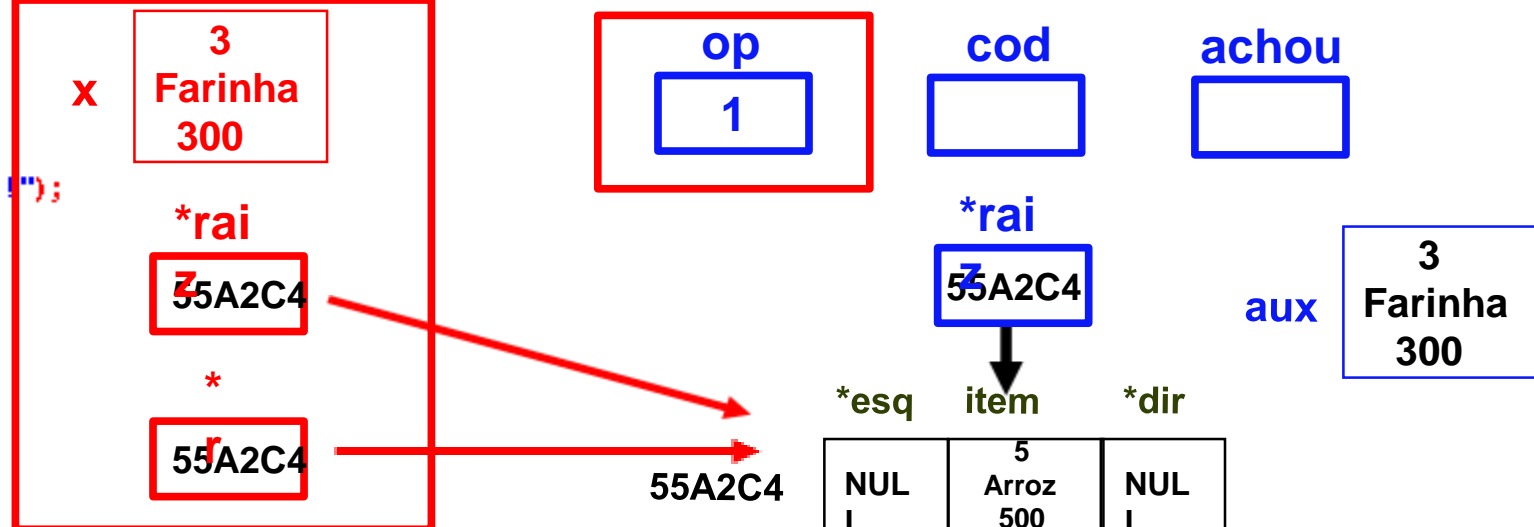
```
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
```

```
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
```

```
    getch(); // segura a tela
```

```
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

```
}
```

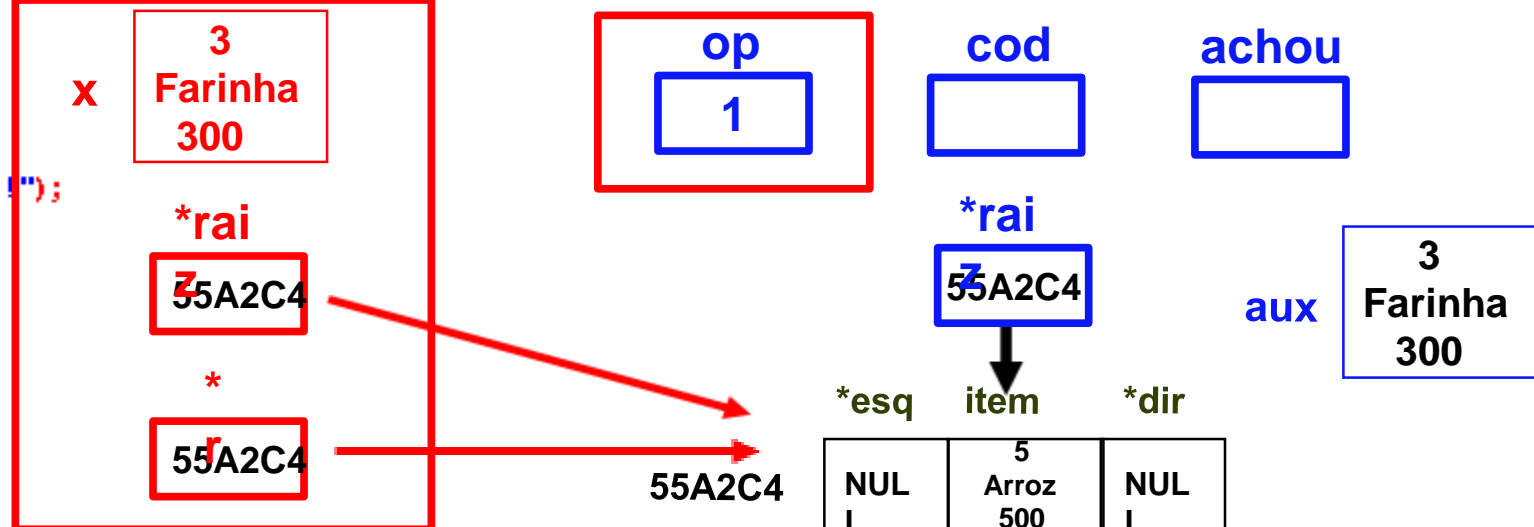
```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
```

```
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

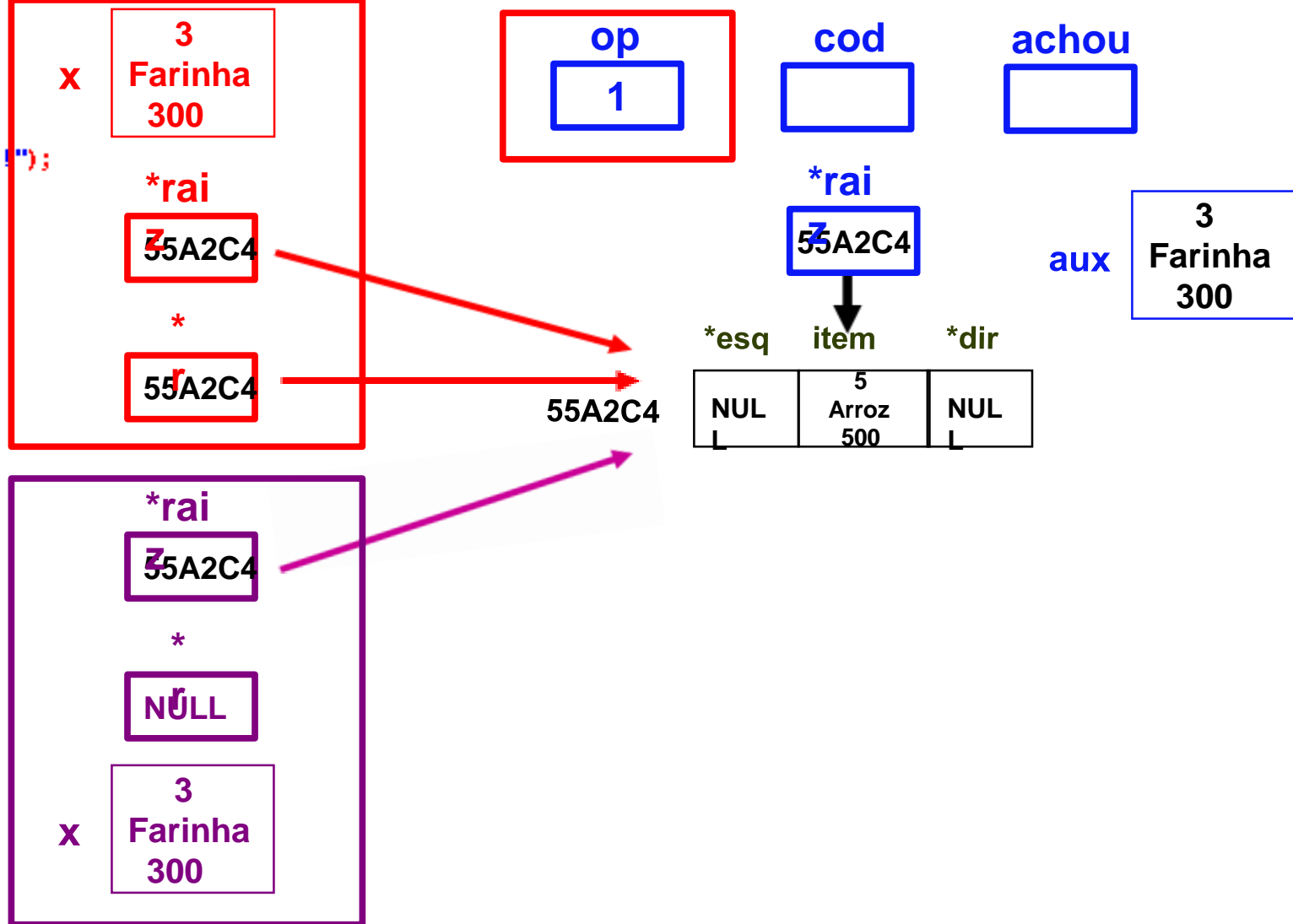
```
return r;
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
    if(!raiz) raiz = inserir(aux, raiz, raiz);
    else inserir(aux, raiz, raiz);
    break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
    if(!r){
        r = (tipo_no *) malloc(sizeof(tipo_no));
        if(!r){
            printf("\n\n ERRO: Problema de espaço na memória !!!");
            getch(); // segura a tela
            exit(0); // fecha o programa
        }
    }
```

```
    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;
```

```
    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");
    getch();
```

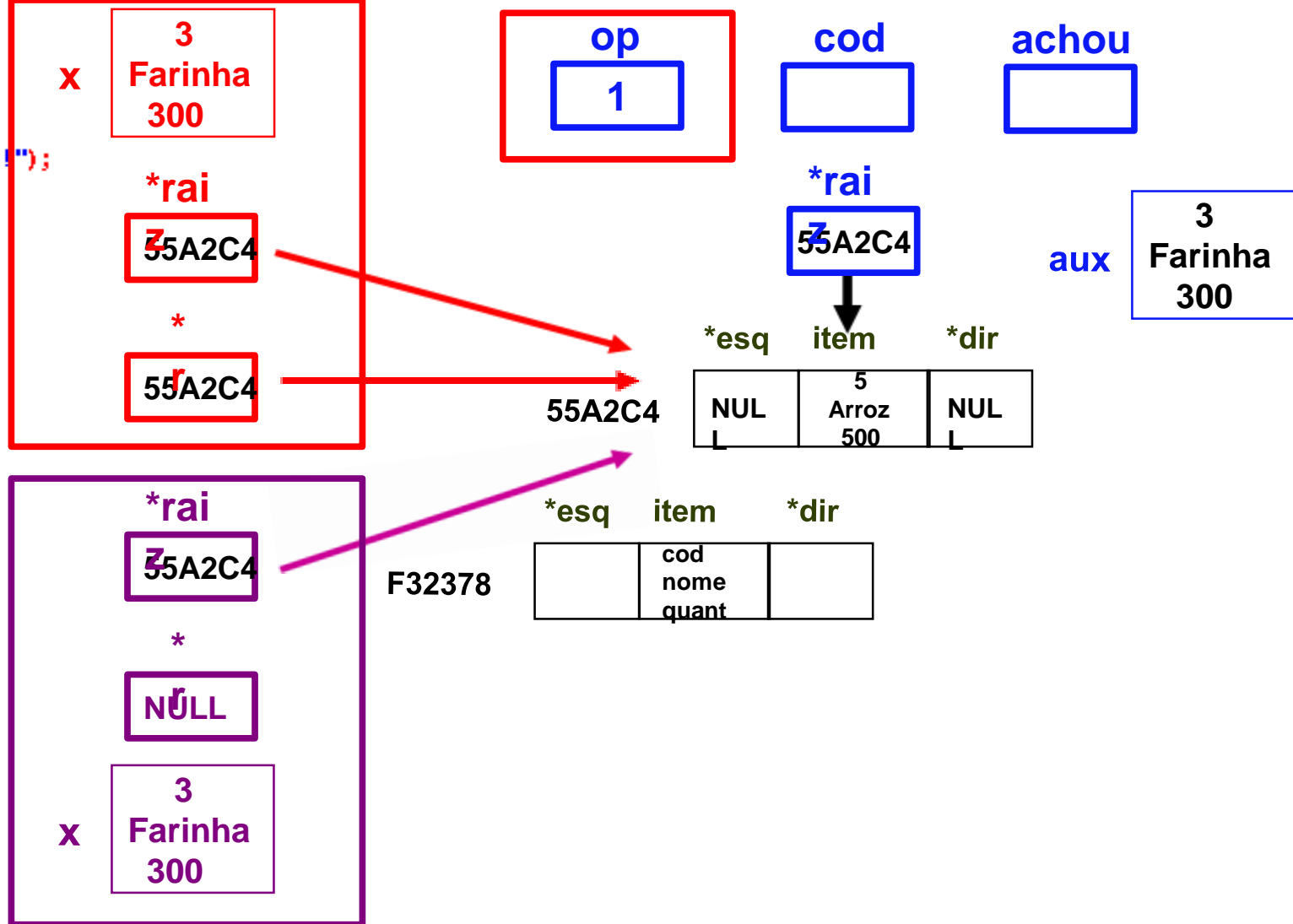
```
    return r;
```

```
    if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
    else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
    else{
        printf("\n\n IMPOSSÍVEL INSERIR !!!");
        printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
        getch(); // segura a tela
    }
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

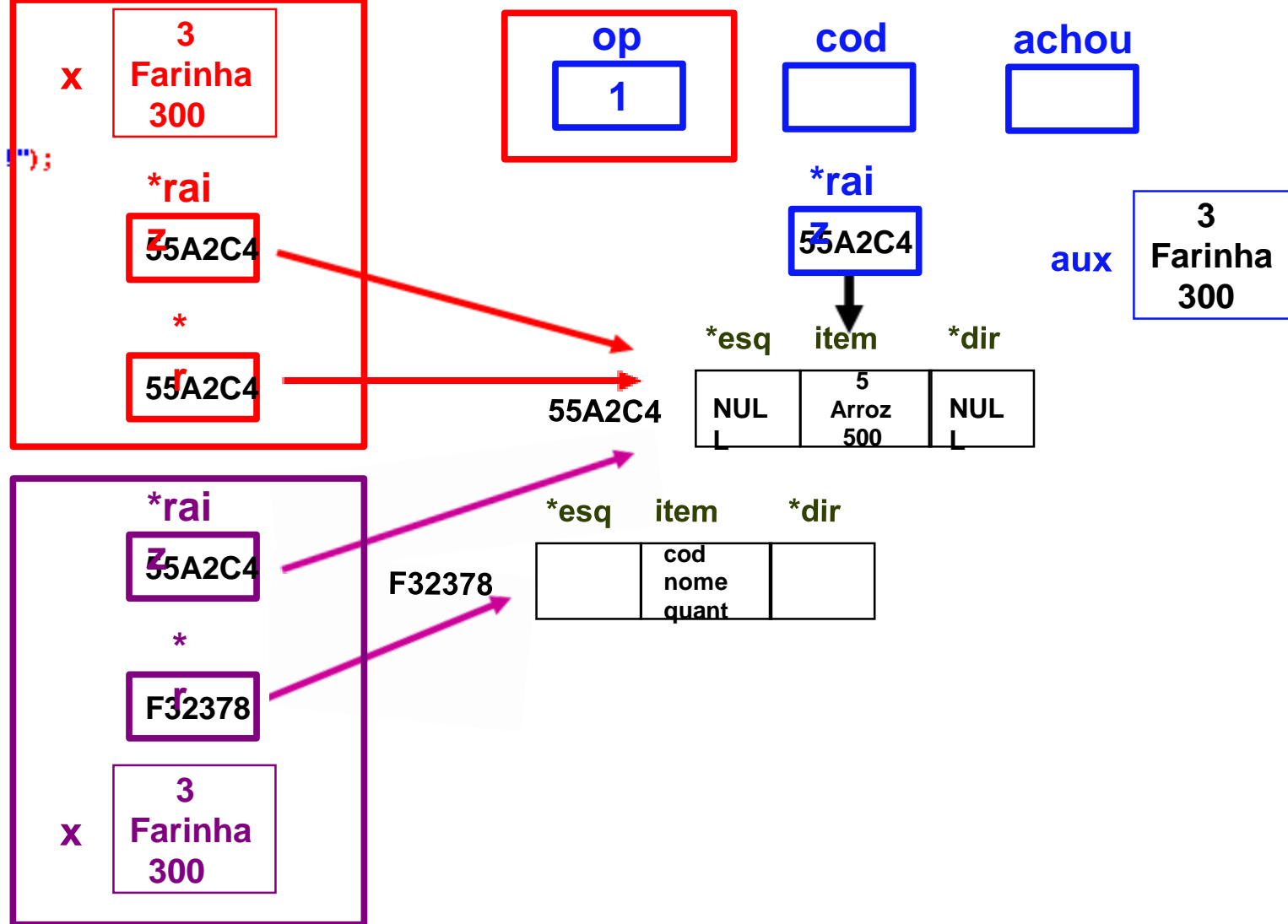
```
return r;
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

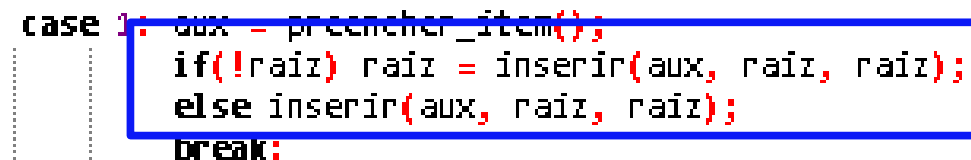
```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
} // fim inserir()
```



```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

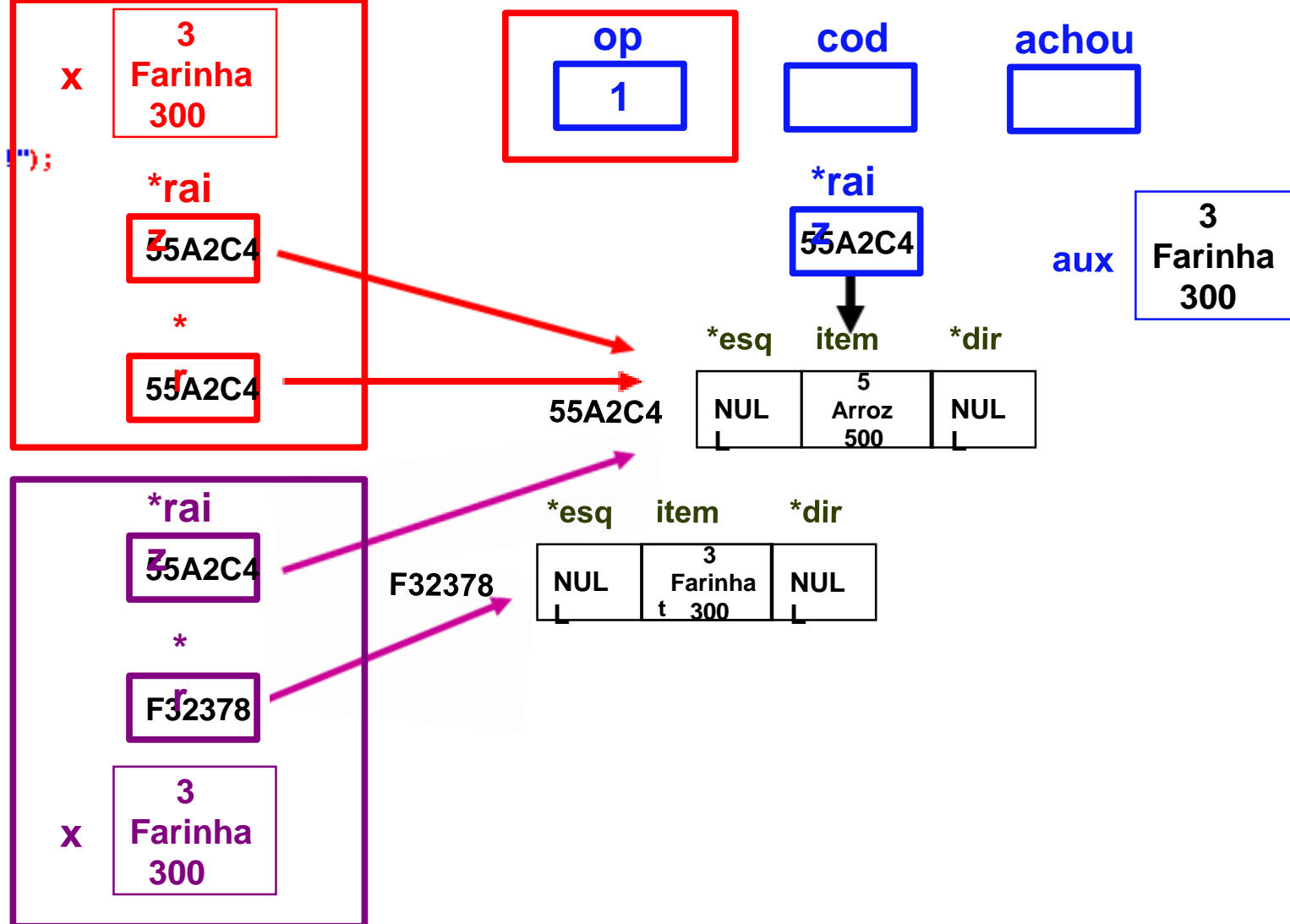
```
return r;
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

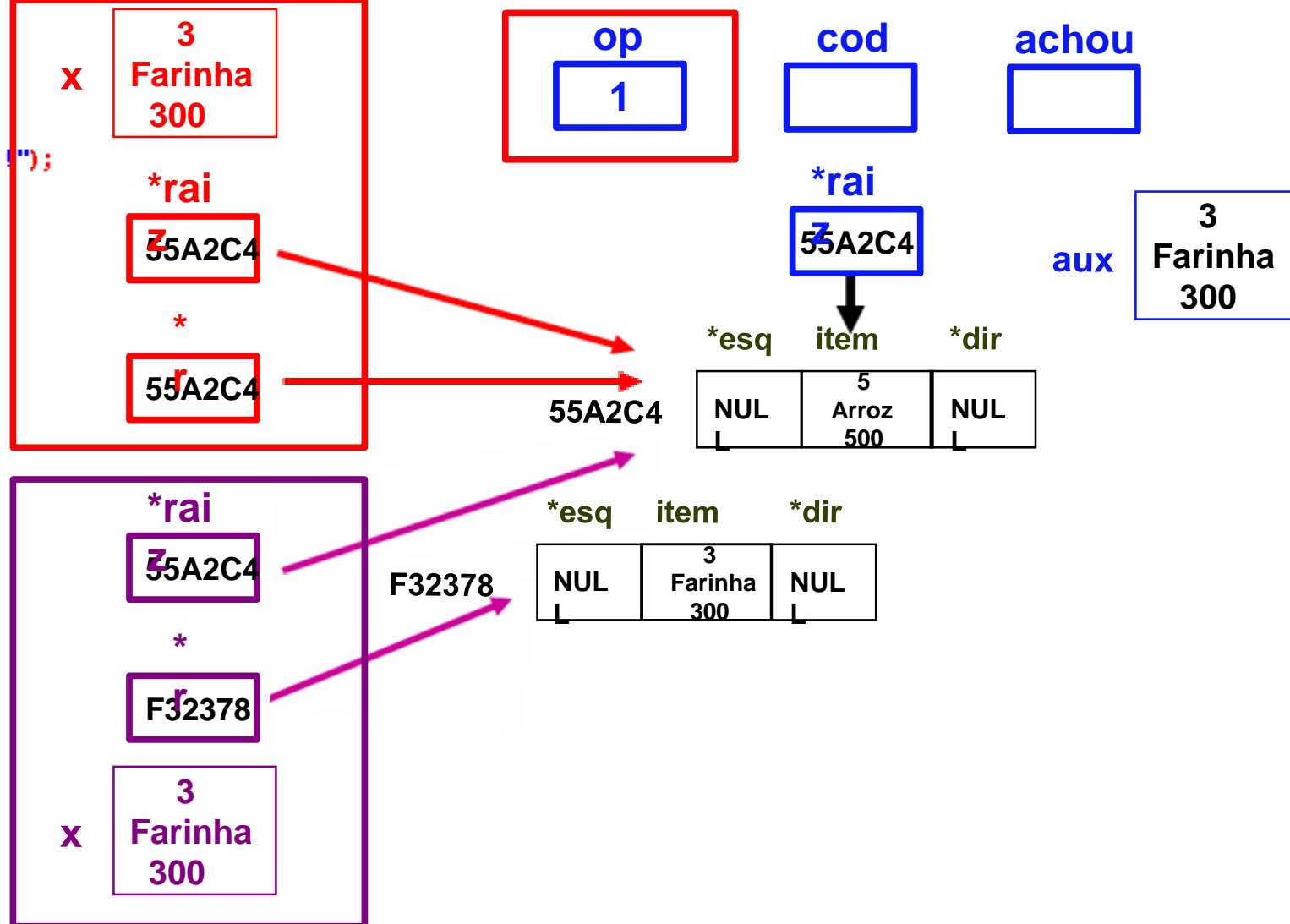
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```




```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

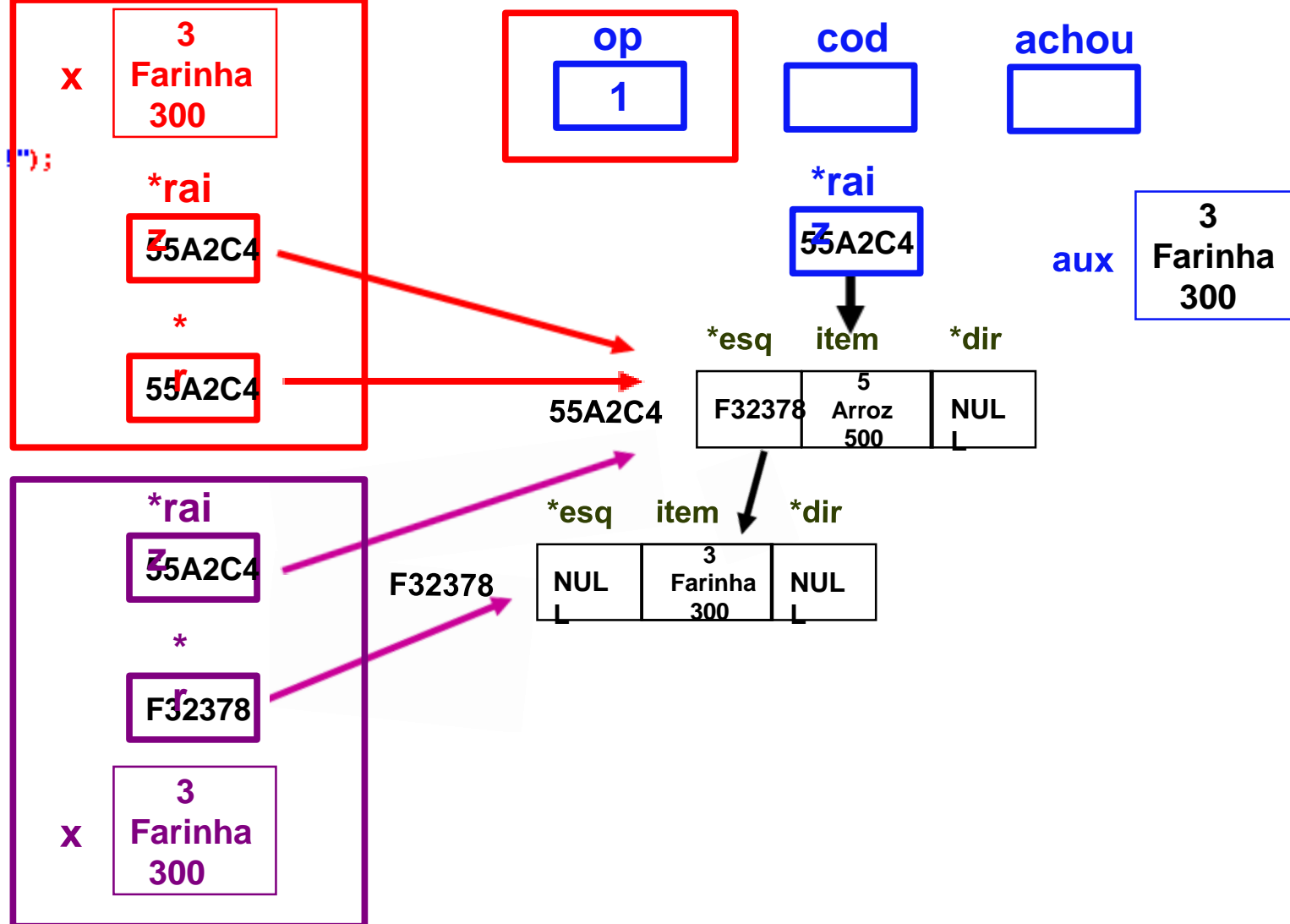
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();

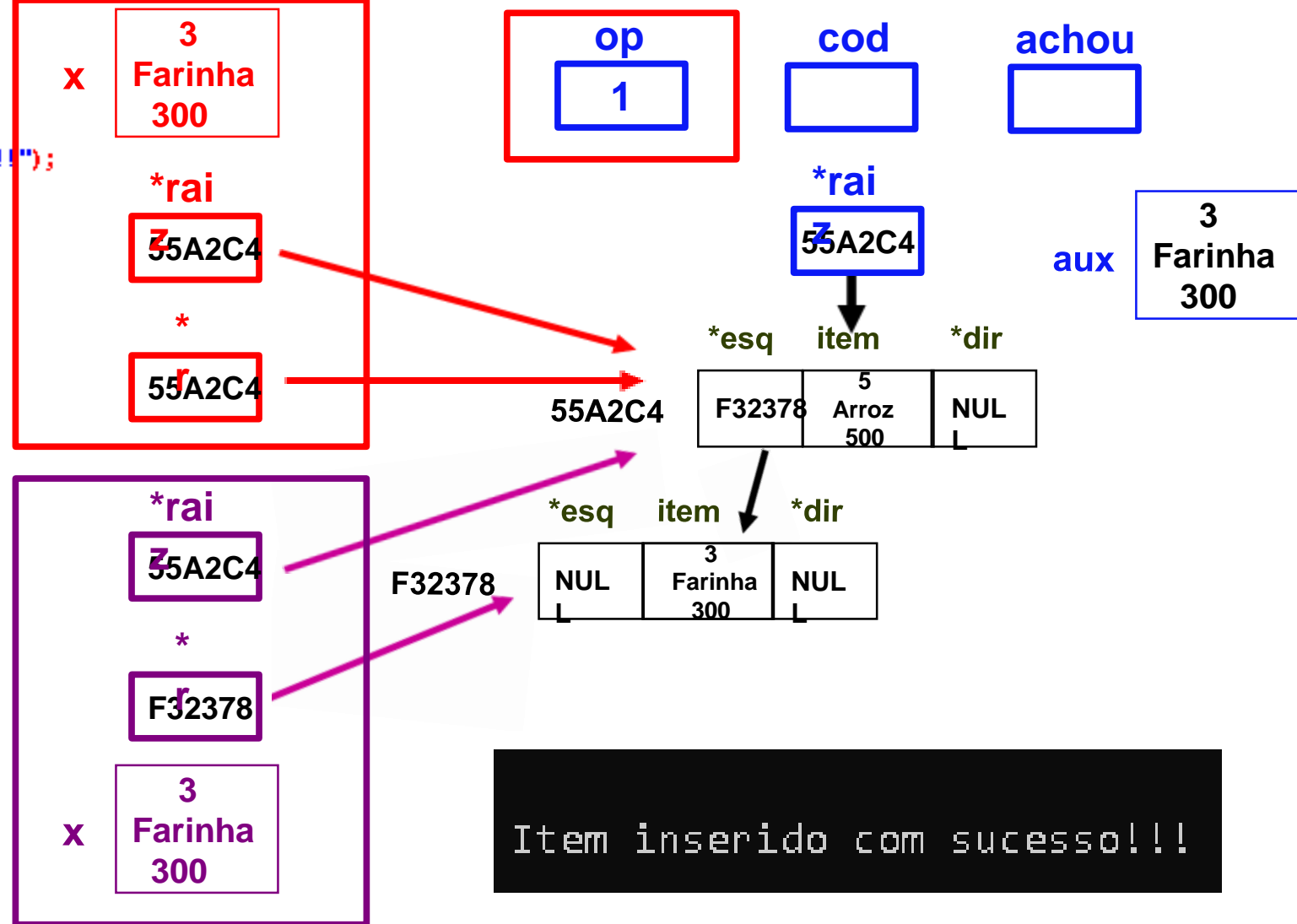
return r;
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

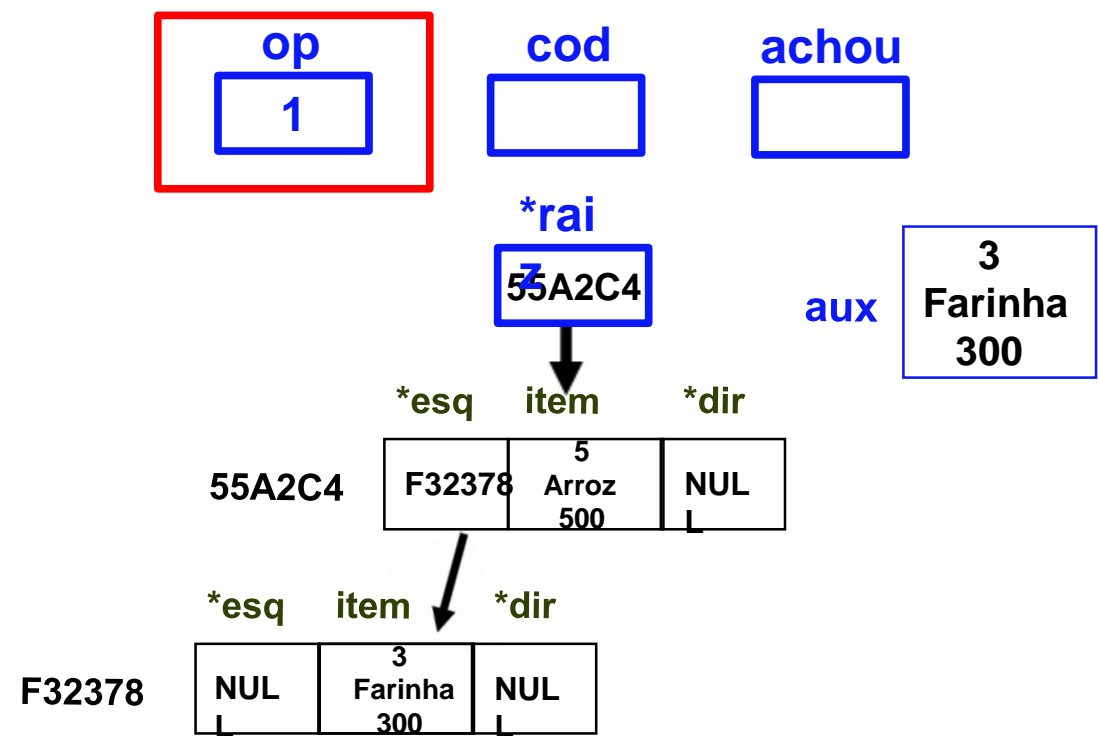
```
} // fim inserir()
```



Item inserido com sucesso!!!

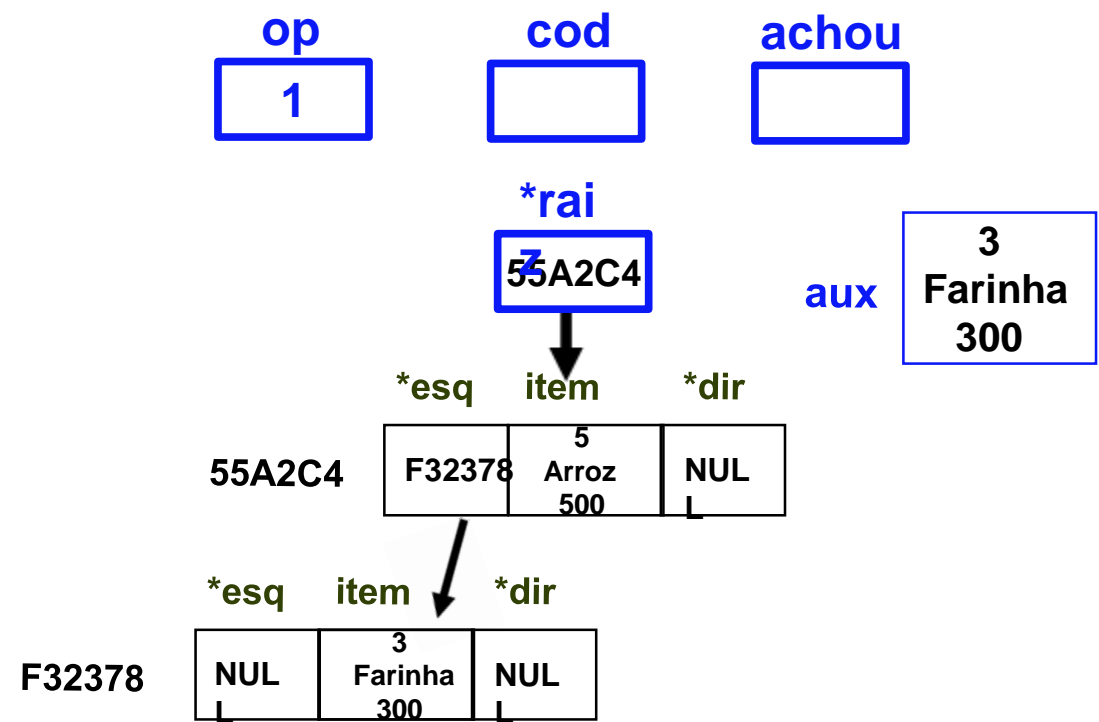
```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

ÁRVORES BINÁRIAS

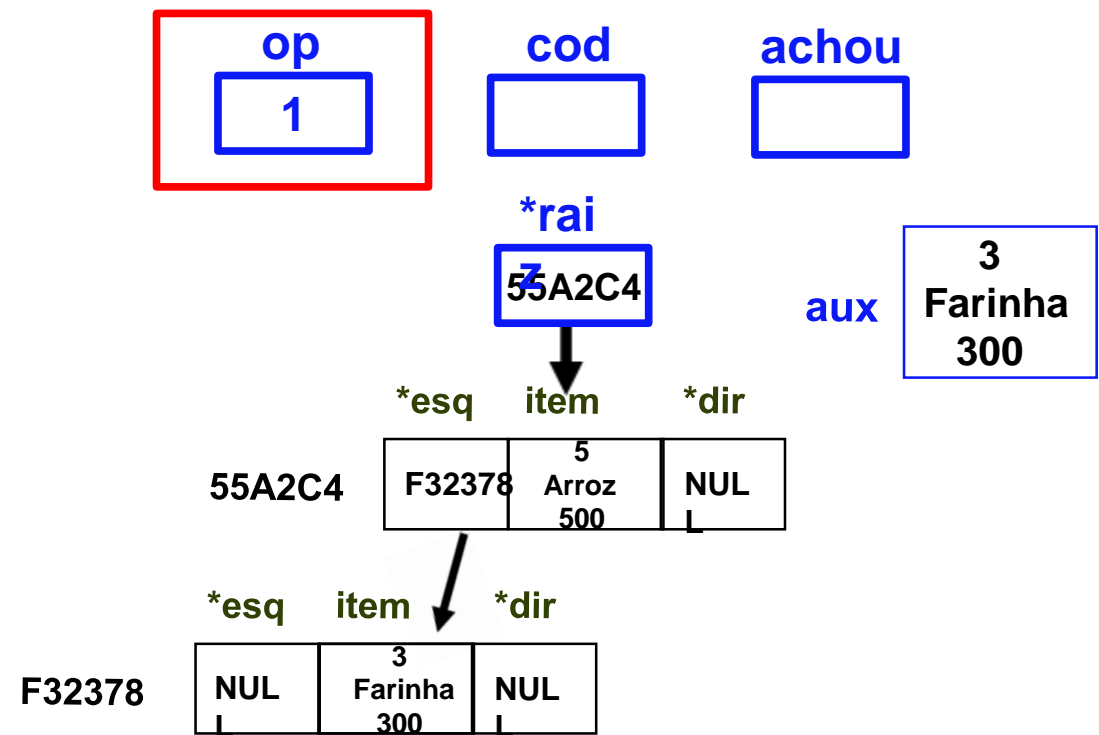


```
case 1: aux = preencher_item();  
        if(!raiz) raiz = inserir(aux, raiz, raiz);  
        else inserir(aux, raiz, raiz);  
        break;
```

ÁRVORES BINÁRIAS



ÁRVORES BINÁRIAS



```
***** MENU *****
[1] - INSERIR
[2] - MOSTRAR
[3] - PESQUISAR
[4] - EXCLUIR
[0] - SAIR
```

Digite sua opção: 1

```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

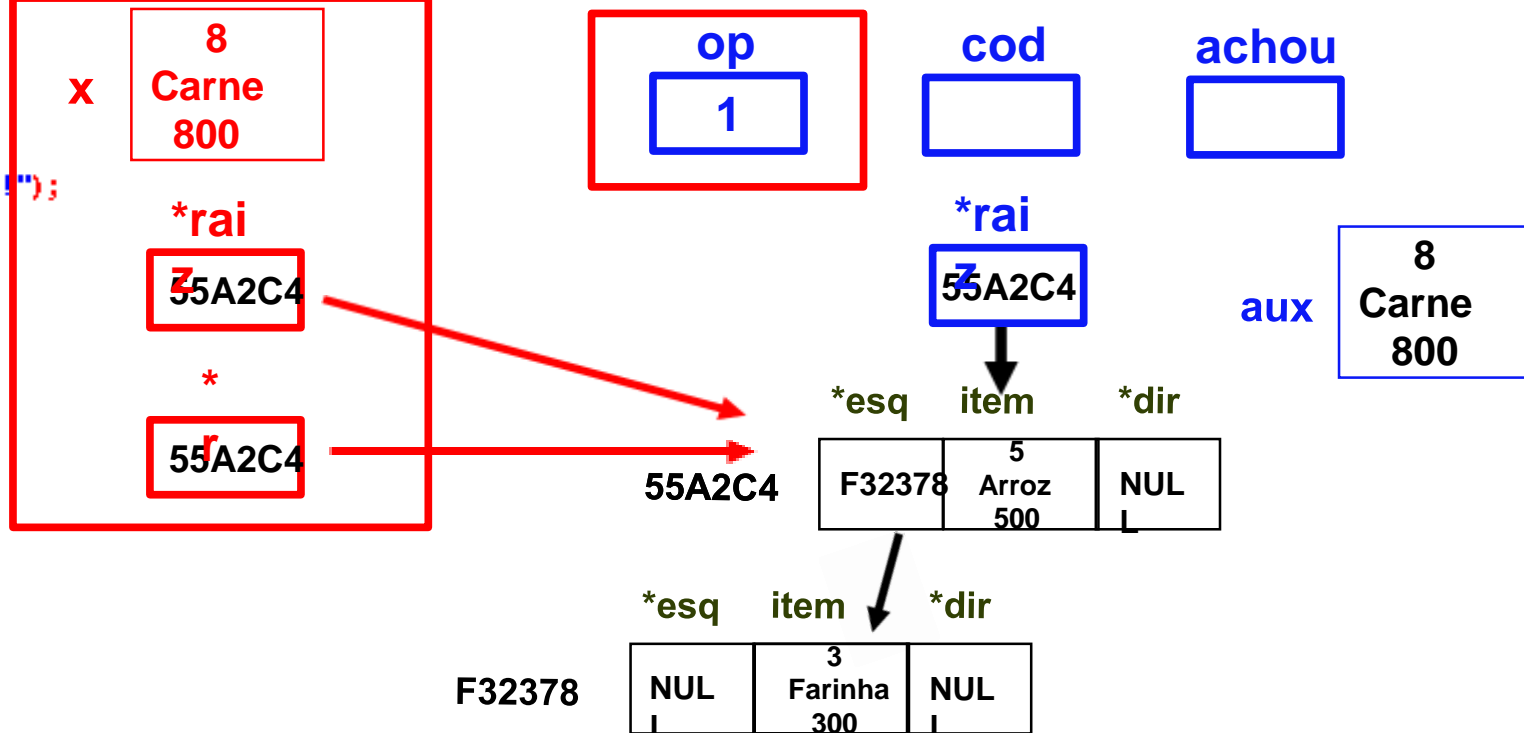
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
    if(!raiz) raiz = inserir(aux, raiz, raiz);
    else inserir(aux, raiz, raiz);
    break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

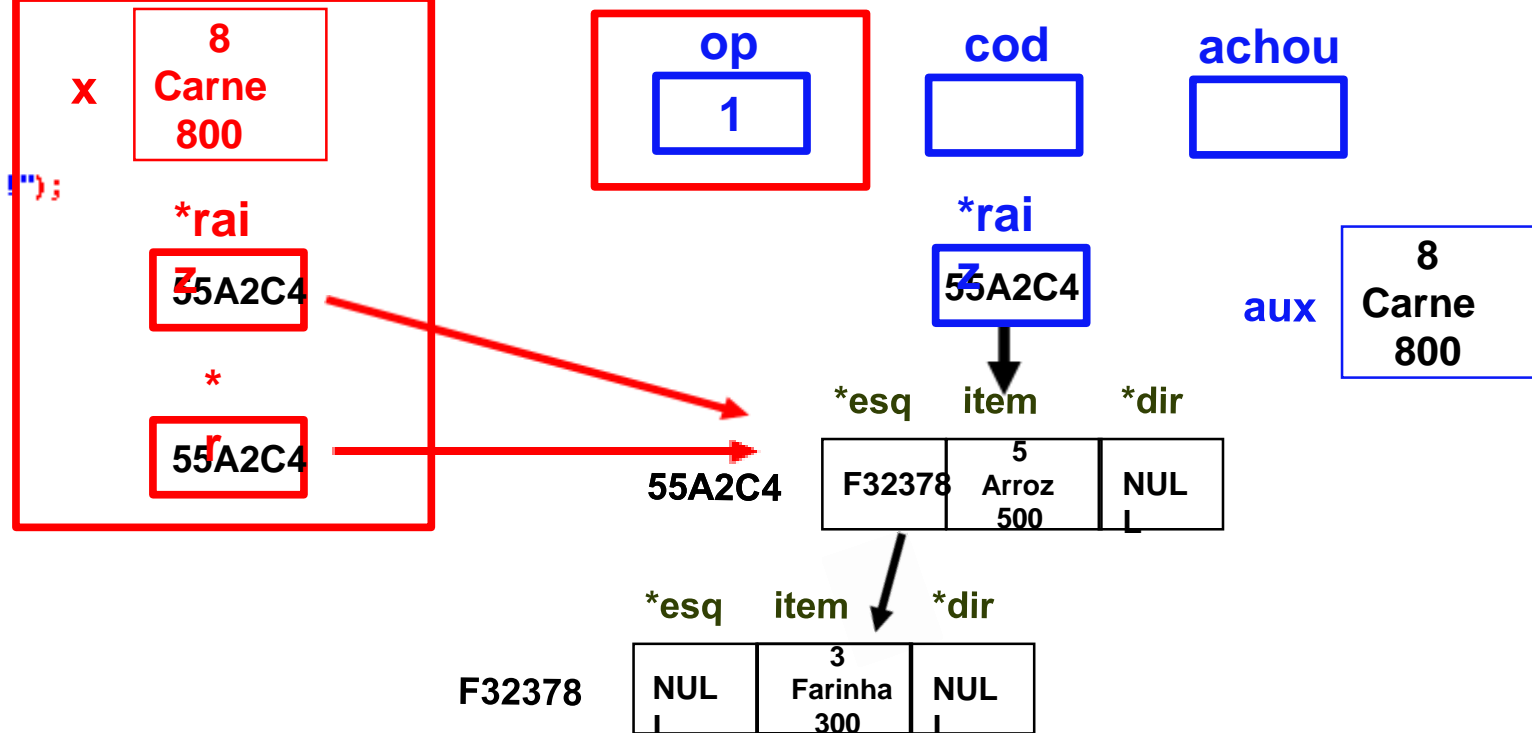
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

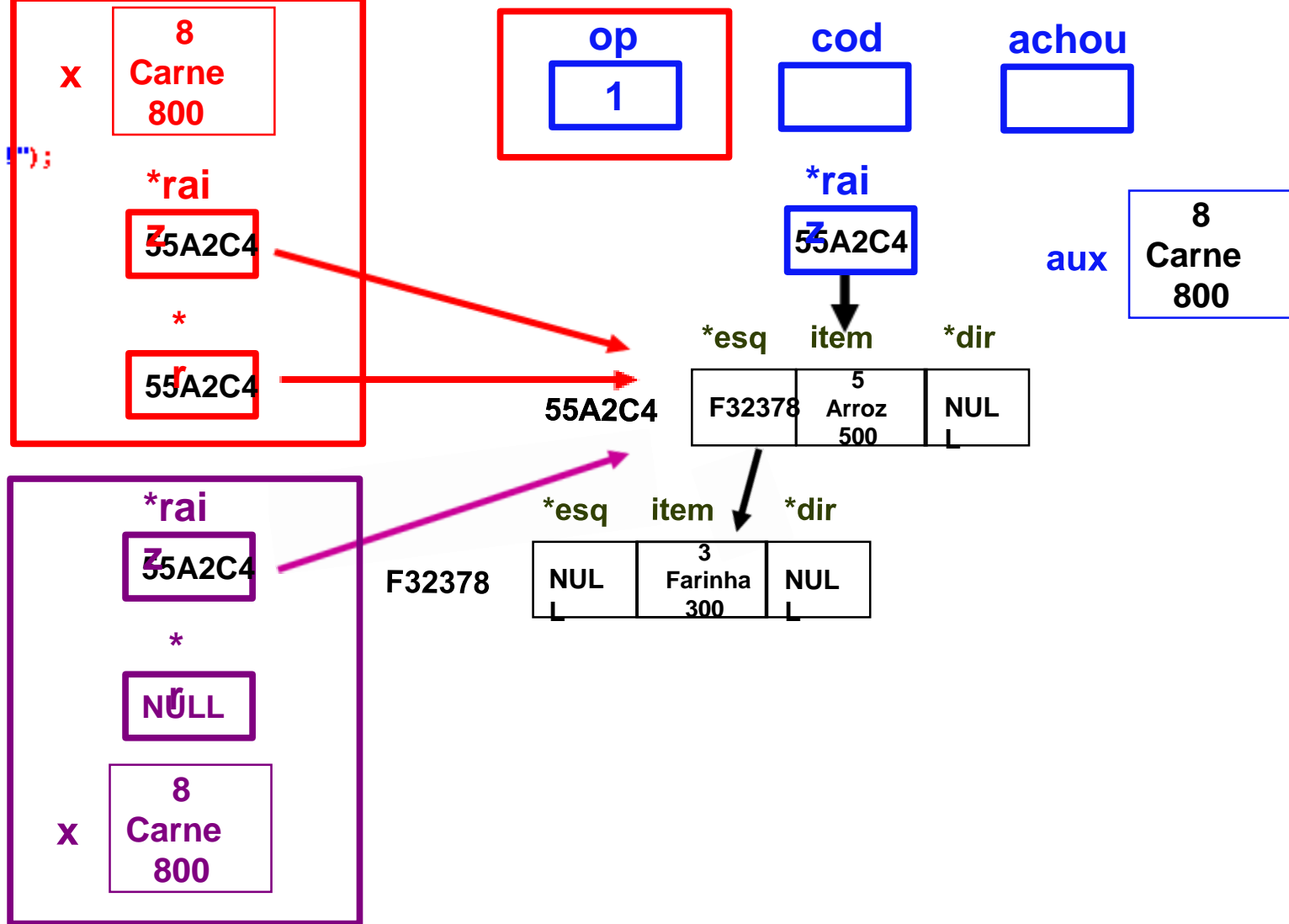
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```



```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){  
    r = (tipo_no *) malloc(sizeof(tipo_no));  
    if(!r){
```

```
        printf("\n\n ERRO: Problema de espaço na memória !!!");  
        getch(); // segura a tela  
        exit(0); // fecha o programa  
    }
```

```
    r->esq = NULL;  
    r->dir = NULL;  
    //r->esq = r->dir = NULL; // atribuição múltipla  
    r->item = x;
```

```
    if(!raiz){  
        printf("\n\n Item inserido com sucesso!!!");  
        getch();  
        return r; // primeira entrada => raiz da árvore  
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");  
    getch();
```

```
    return r;
```

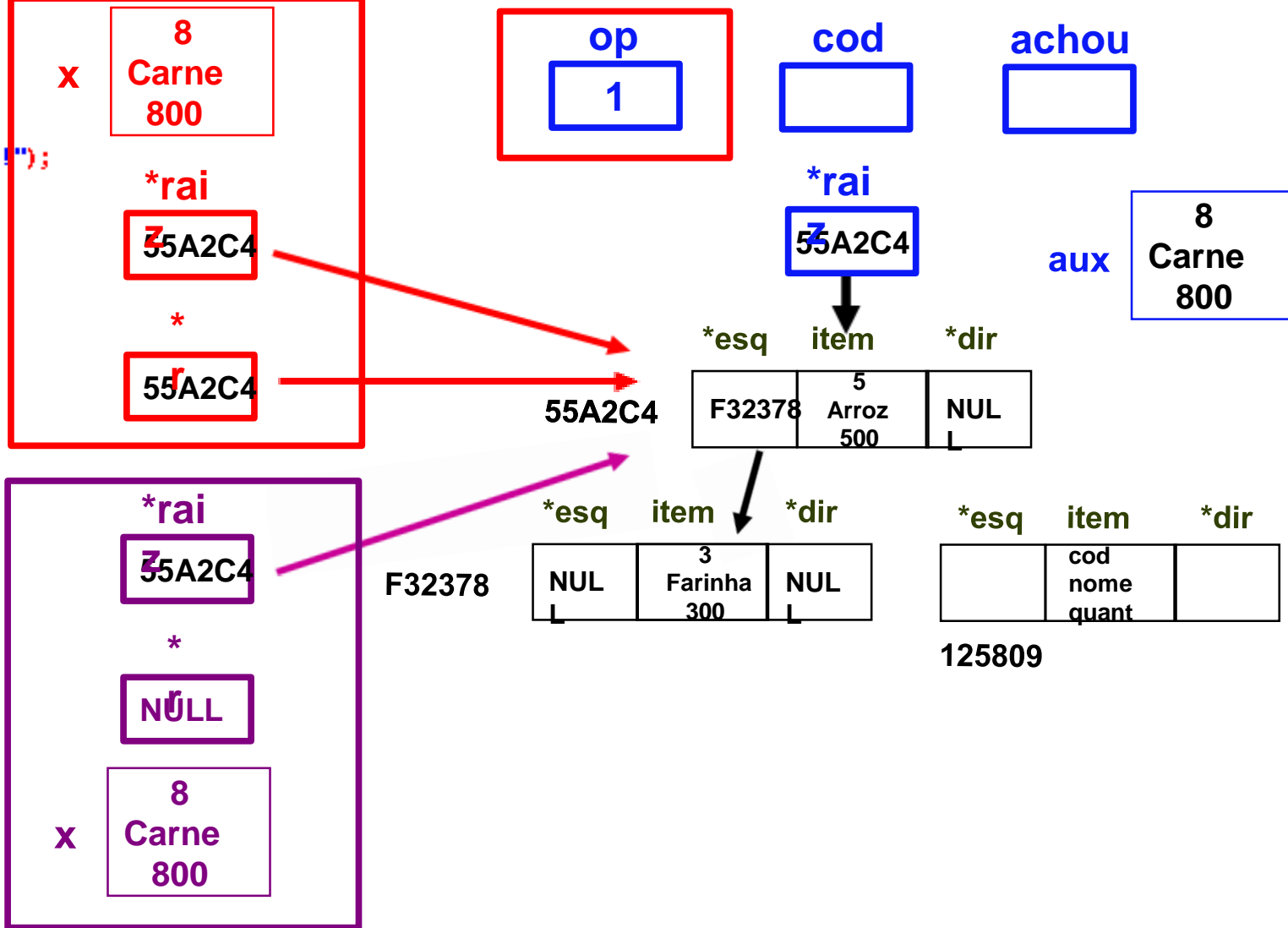
```
}
```

```
if(x.cod < r->item.cod) inserir(x , r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x , r, r->dir);
```

```
else{  
    printf("\n\n IMPOSSÍVEL INSERIR !!!");  
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);  
    getch(); // segura a tela  
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();  
if(!raiz) raiz = inserir(aux, raiz, raiz);  
else inserir(aux, raiz, raiz);  
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){  
    r = (tipo_no *) malloc(sizeof(tipo_no));  
    if(!r){
```

```
        printf("\n\n ERRO: Problema de espaço na memória !!!");  
        getch(); // segura a tela  
        exit(0); // fecha o programa  
    }
```

```
    r->esq = NULL;  
    r->dir = NULL;  
    //r->esq = r->dir = NULL; // atribuição múltipla  
    r->item = x;
```

```
    if(!raiz){  
        printf("\n\n Item inserido com sucesso!!!");  
        getch();  
        return r; // primeira entrada => raiz da árvore  
    }
```

```
    if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
    else raiz->dir = r;
```

```
    printf("\n\n Item inserido com sucesso!!!");  
    getch();
```

```
    return r;
```

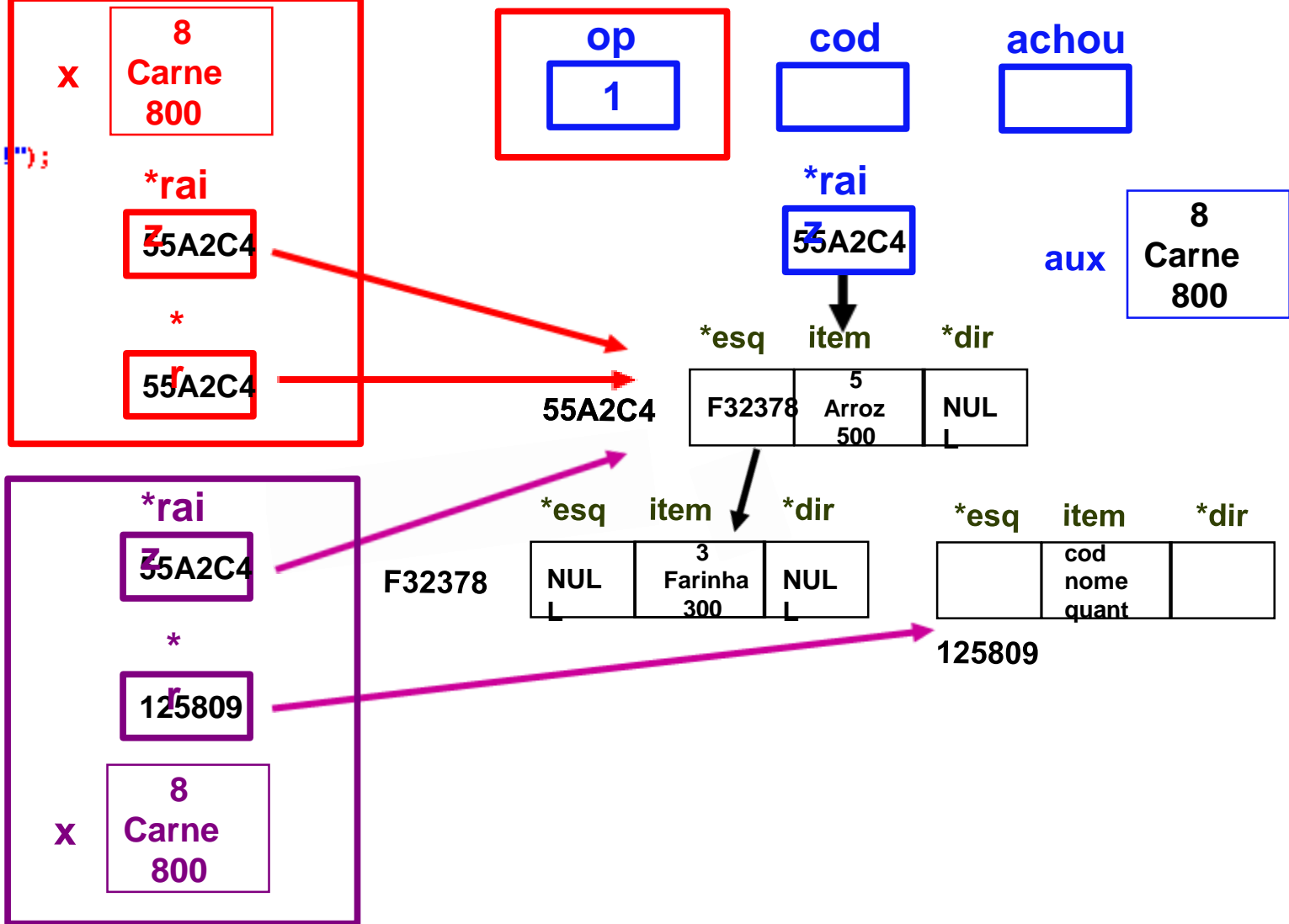
```
}
```

```
if(x.cod < r->item.cod) inserir(x , r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x , r, r->dir);
```

```
else{  
    printf("\n\n IMPOSSÍVEL INSERIR !!!");  
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);  
    getch(); // segura a tela  
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();  
if(!raiz) raiz = inserir(aux, raiz, raiz);  
else inserir(aux, raiz, raiz);  
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

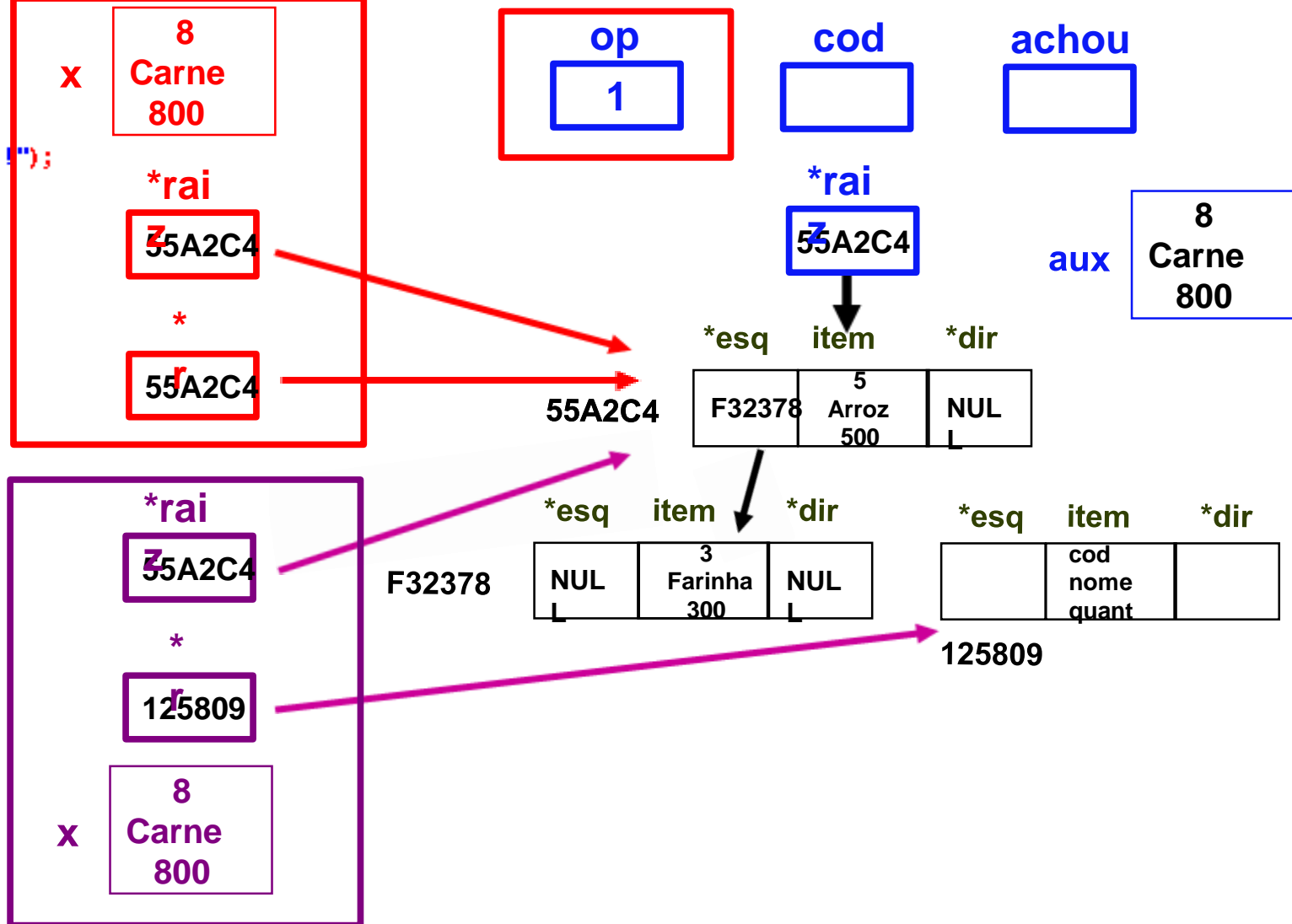
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
    if(!raiz) raiz = inserir(aux, raiz, raiz);
    else inserir(aux, raiz, raiz);
    break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

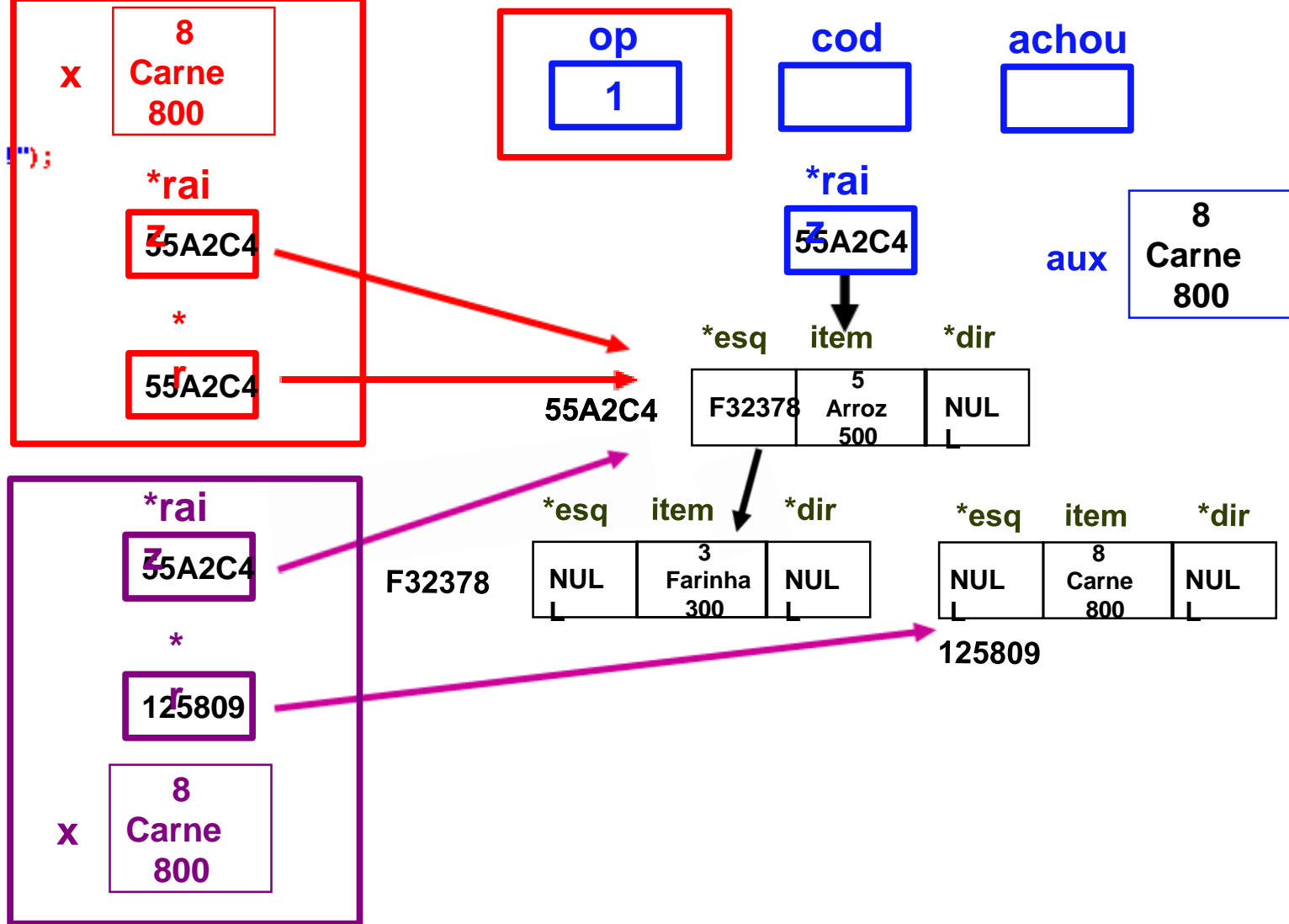
```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
```

```
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }

    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;

    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
else raiz->dir = r;

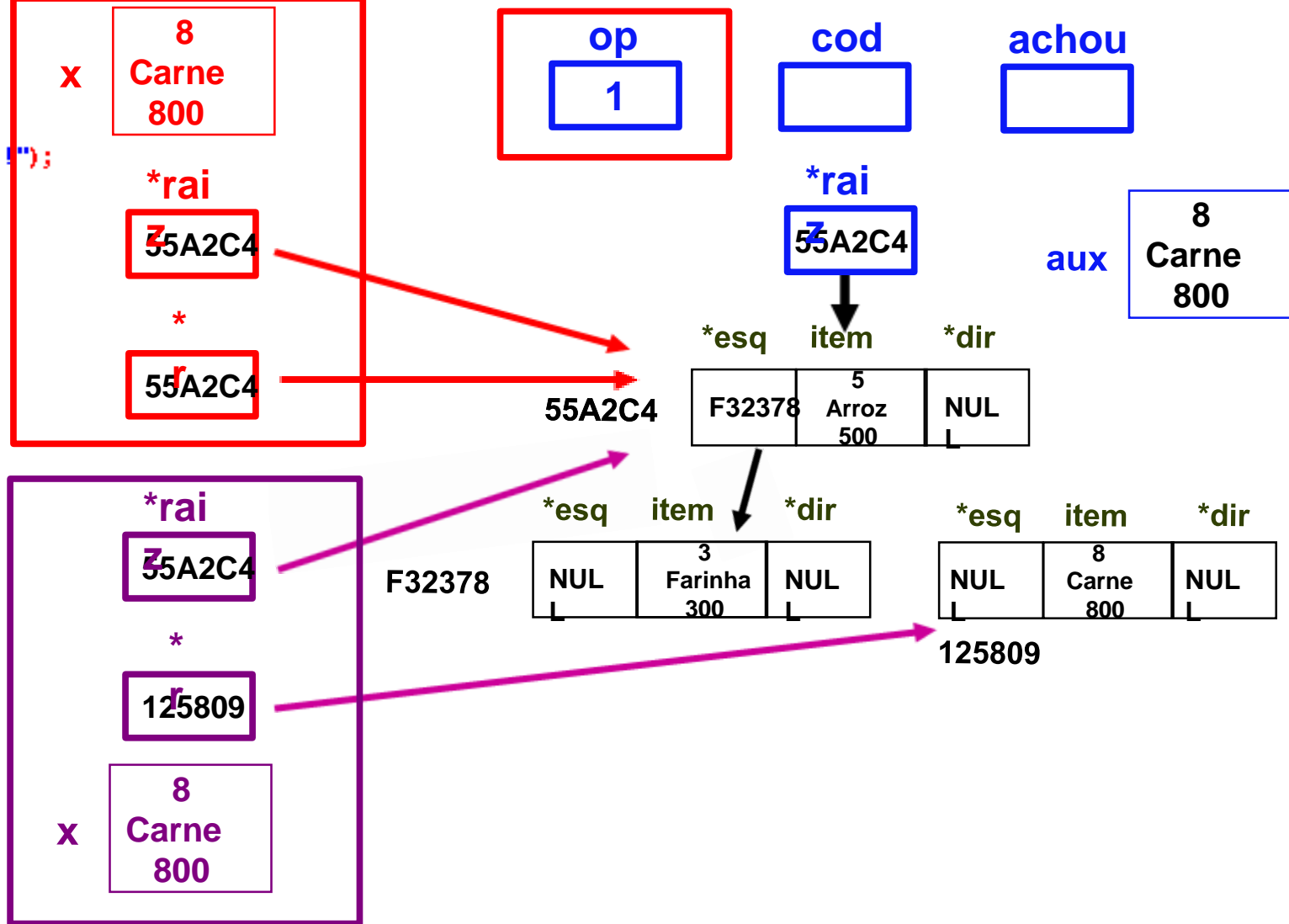
printf("\n\n Item inserido com sucesso!!!");
getch();

return r;
}
```

```
if(x.cod < r->item.cod) inserir(x , r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x , r, r->dir);
```

```
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
} // fim inserir()
```



```
case 1: aux = preencher_item();
if(!raiz) raiz = inserir(aux, raiz, raiz);
else inserir(aux, raiz, raiz);
break;
```

```

if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }

    r->esq = NULL;
    r->dir = NULL;
    //r->esq = r->dir = NULL; // atribuição múltipla
    r->item = x;

    if(!raiz){
        printf("\n\n Item inserido com sucesso!!!");
        getch();
        return r; // primeira entrada => raiz da árvore
    }

    if(x.cod < raiz->item.cod) raiz->esq = r;
    else raiz->dir = r;

    printf("\n\n Item inserido com sucesso!!!");
    getch();

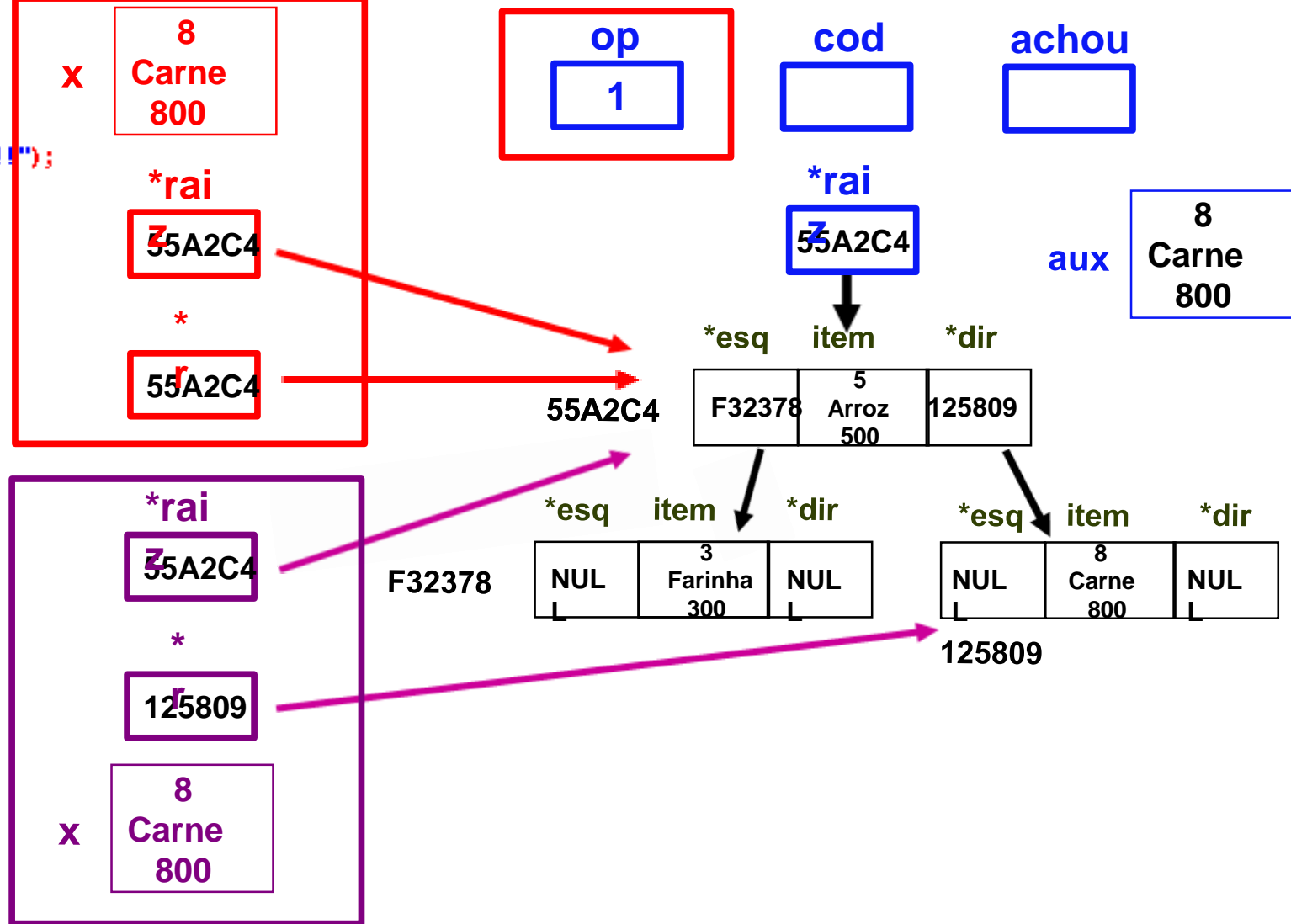
    return r;
}

if(x.cod < r->item.cod) inserir(x, r, r->esq);
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
else{
    printf("\n\n IMPOSSÍVEL INSERIR !!!");
    printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
    getch(); // segura a tela
}
// fim inserir()

```

Diagram Details:

- Root Node:** *raiz (Z55A2C4)
- Left Child:** 55A2C4 (Carne 800)
- Right Child:** * (125809)
- Right-Right Child:** 8 (Carne 800)



```
case 1: aux = prechener_item();  
        if(!raiz) raiz = inserir(aux, raiz, raiz);  
        else inserir(aux, raiz, raiz);  
        break;
```

```
tipo_no* inserir(tipo_item x, tipo_no *raiz, tipo_no *r){
```

```
if(!r){
    r = (tipo_no *) malloc(sizeof(tipo_no));
    if(!r){
        printf("\n\n ERRO: Problema de espaço na memória !!!");
        getch(); // segura a tela
        exit(0); // fecha o programa
    }
}
```

```
r->esq = NULL;
r->dir = NULL;
//r->esq = r->dir = NULL; // atribuição múltipla
r->item = x;
```

```
if(!raiz){
    printf("\n\n Item inserido com sucesso!!!");
    getch();
    return r; // primeira entrada => raiz da árvore
}
```

```
if(x.cod < raiz->item.cod) raiz->esq = r;
```

```
else raiz->dir = r;
```

```
printf("\n\n Item inserido com sucesso!!!");
getch();
```

```
return r;
```

```
}
```

```
if(x.cod < r->item.cod) inserir(x, r, r->esq);
```

```
else if(x.cod > r->item.cod) inserir(x, r, r->dir);
```

```
else{
```

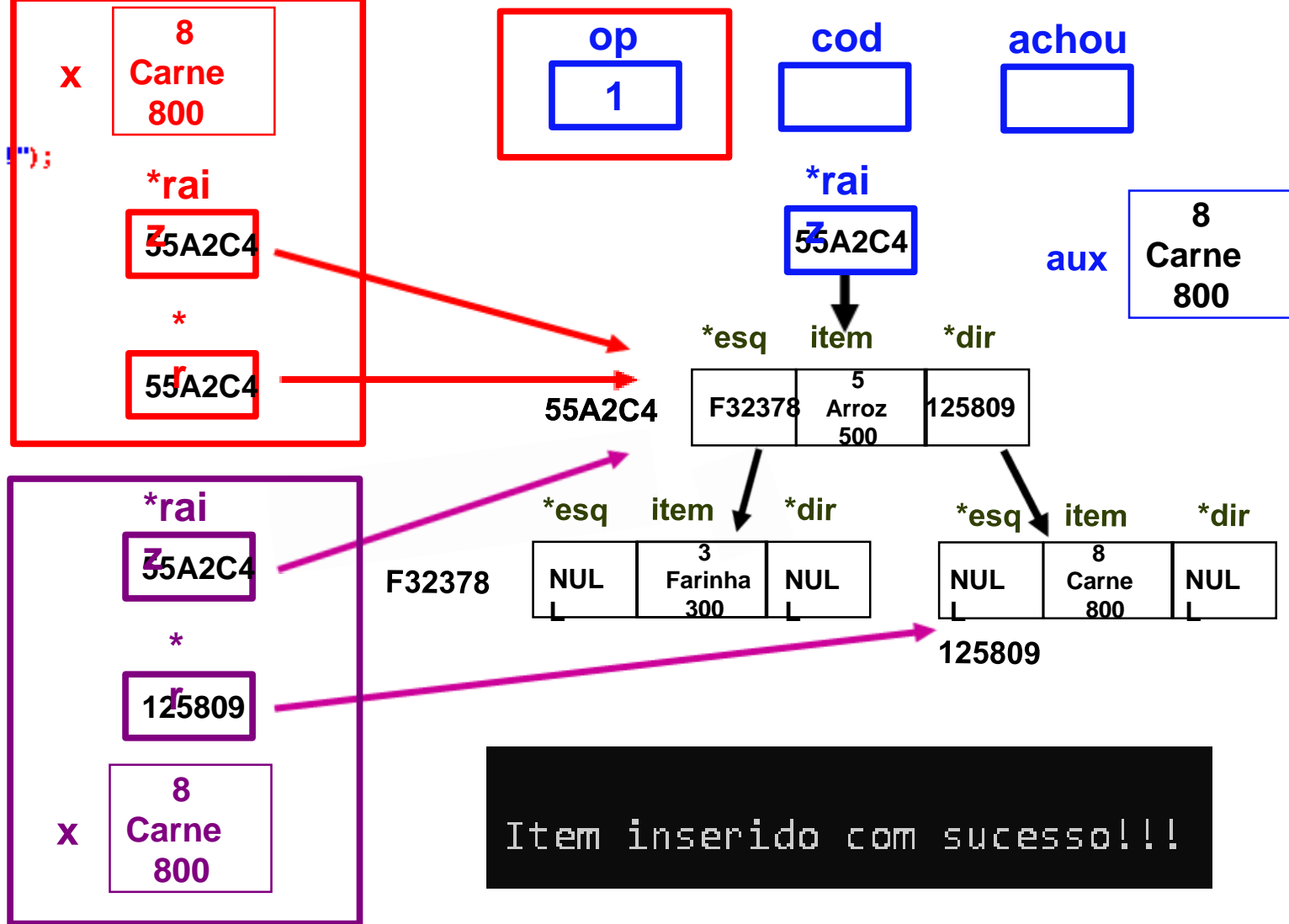
```
printf("\n\n IMPOSSÍVEL INSERIR !!!");
```

```
printf("\n\n MOTIVO: Já existe um item com o código %d na árvore", x.cod);
```

```
getch(); // segura a tela
```

```
}
```

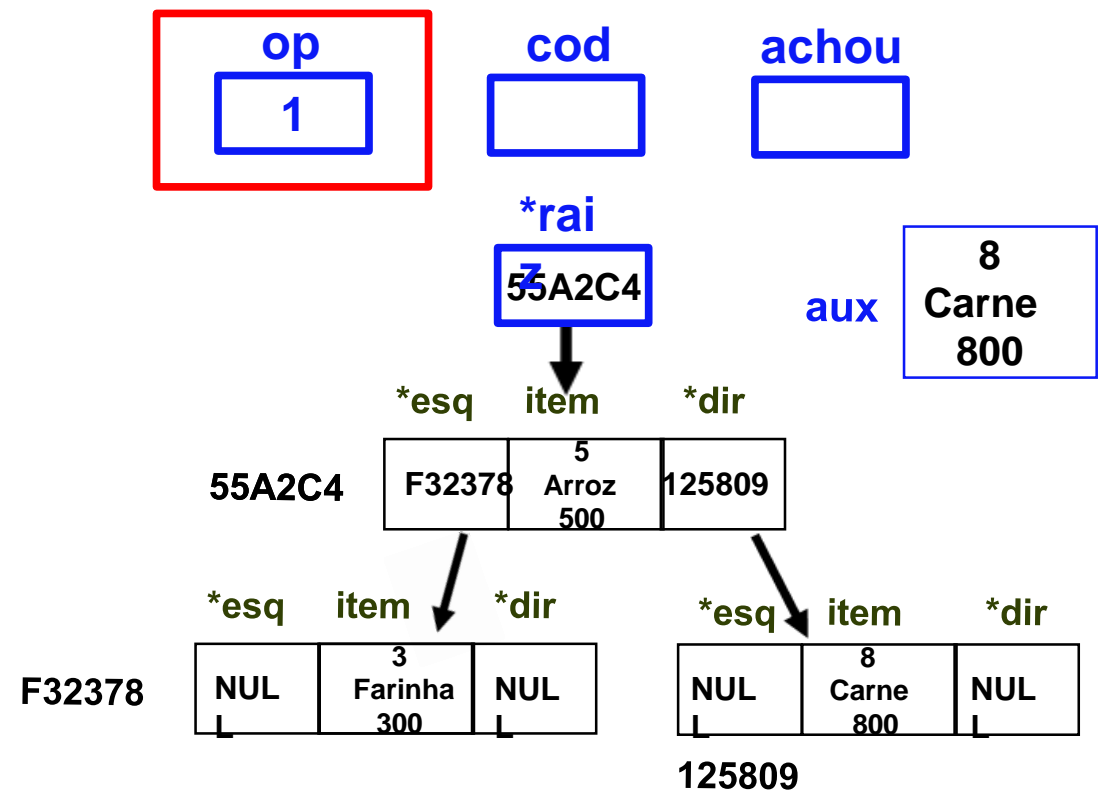
```
} // fim inserir()
```



Item inserido com sucesso!!!

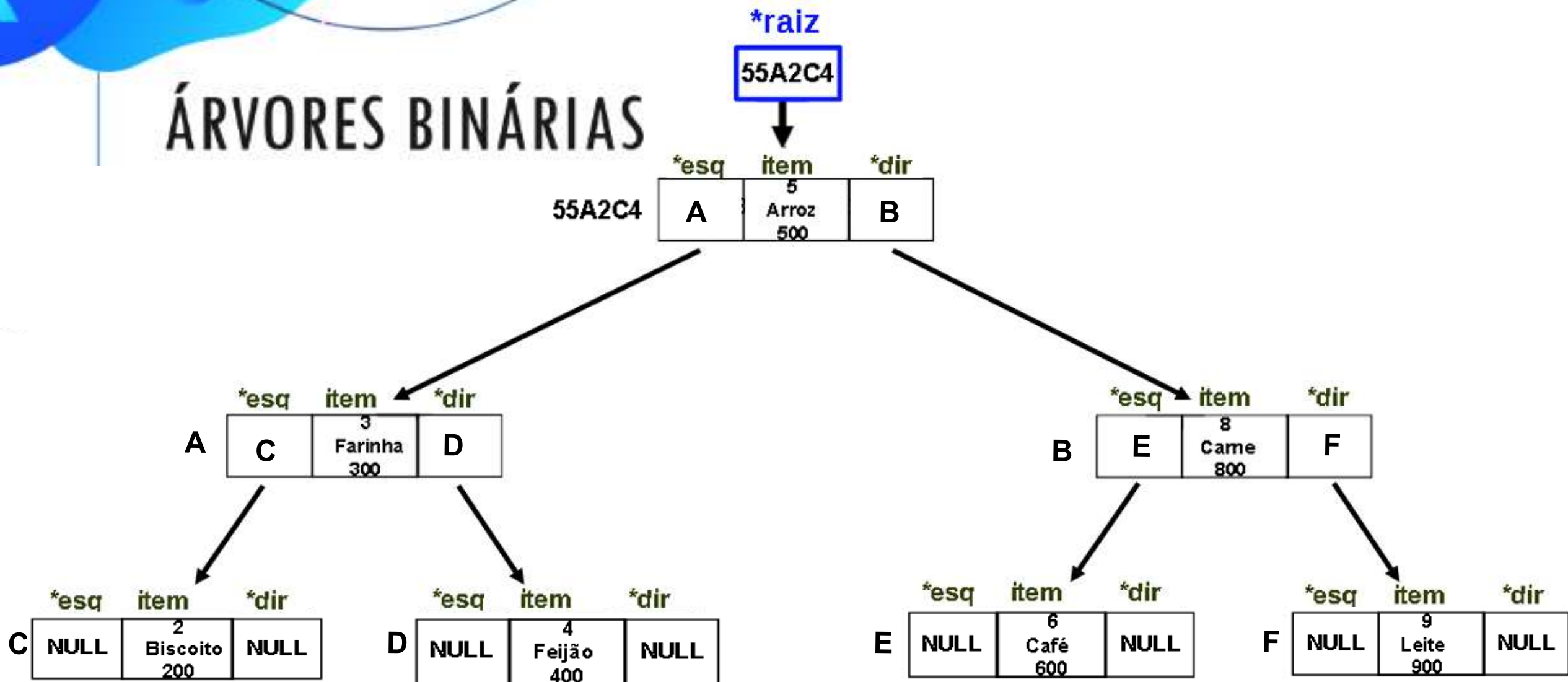
```
case 1: aux = preencher_item();
        if(!raiz) raiz = inserir(aux, raiz, raiz);
        else inserir(aux, raiz, raiz);
        break;
```

ÁRVORES BINÁRIAS



```
case 1: aux = preencher_item();  
        if(!raiz) raiz = inserir(aux, raiz, raiz);  
        else inserir(aux, raiz, raiz);  
        break;
```


ÁRVORES BINÁRIAS





ÁRVORES BINÁRIAS

IMPLEMENTAÇÃO

MOSTRAR O CONTEÚDO DA ÁRVORE

ÁRVORES BINÁRIAS

```
void mostrar(tipo_no *raiz){  
  
    if(!raiz) return;  
  
    mostrar(raiz->esq);  
  
    printf("\n*****");  
  
    printf("\n cod: %d ", raiz->item.cod);  
    printf("\n Nome: %s ", raiz->item.nome);  
    printf("\n Quantidade: %d \n", raiz->item.cod);  
    mostrar(raiz->dir);  
}
```

```
int main() {  
    setlocale(LC_ALL, "");  
  
    int op, cod, achou;  
  
    tipo_no *raiz = NULL;  
    tipo_item aux;
```

```
do{  
    system("cls");  
  
    op = menu();  
  
    switch(op){  
  
        case 0: printf("\n\n\n\t Fim do Programa. \n\n");  
                getch();  
                return(0);  
  
        case 1: aux = preencher_item();  
                if(!raiz) raiz = inserir(aux, raiz, raiz);  
                else inserir(aux, raiz, raiz);  
                break;  
  
        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                mostrar(raiz);  
                getch(); // segura a tela  
                break;  
  
        case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                printf("\n\n Digite o código do item a pesquisar: ");  
                scanf("%d", &cod);  
                achou = 0;  
                pesquisar(&achou, cod, raiz);  
                if(!achou) printf("\n\n O item de código %d não está na árvore", cod);  
                getch(); // segura a tela  
                break;
```

op

2

cod

achou

*rai

55A2C4

aux

cod
nome
quant

```
***** MENU *****  
[1] - INSERIR  
[2] - MOSTRAR  
[3] - PESQUISAR  
[4] - EXCLUIR  
[0] - SAIR
```

Digite sua opção: 2

CONTEÚDO DA ÁRVORE:

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

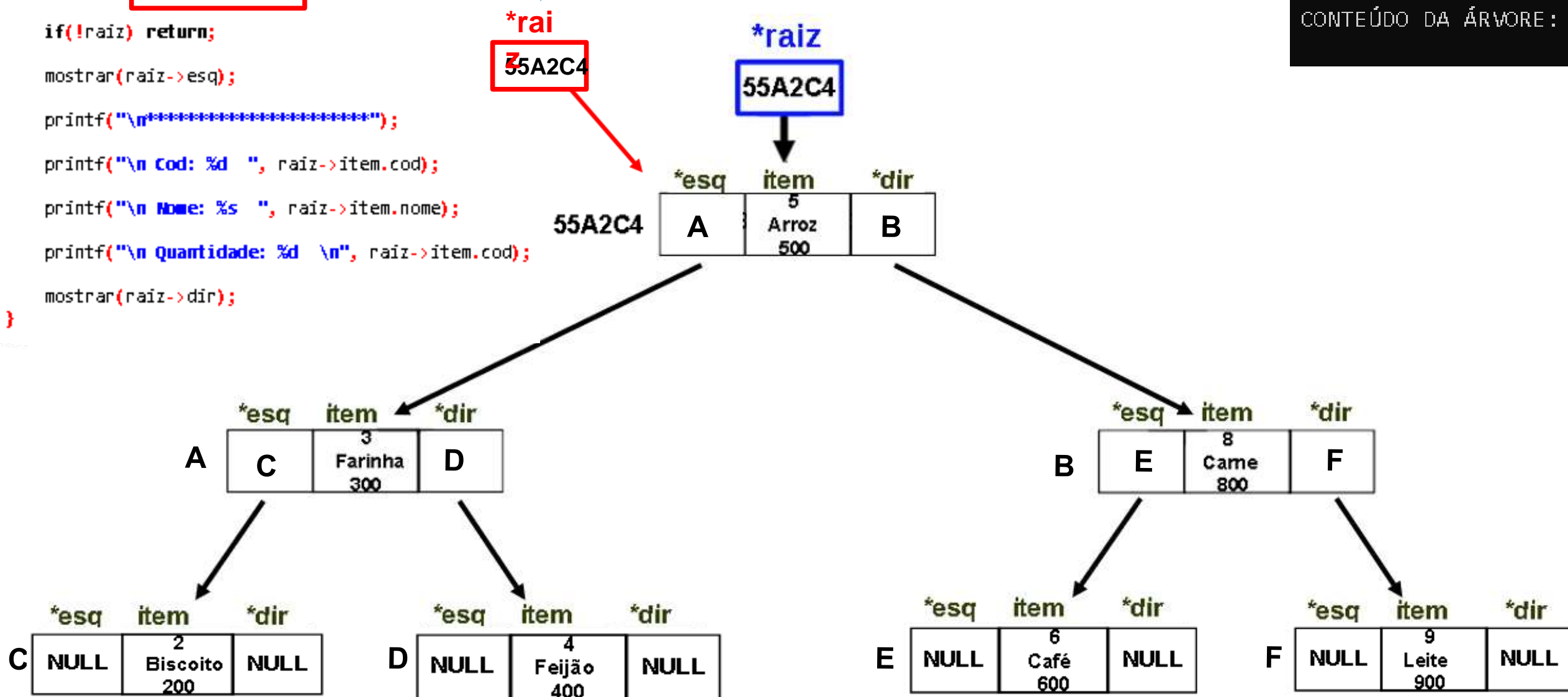
```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

CONTEÚDO DA ÁRVORE :



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

*rai

Z A

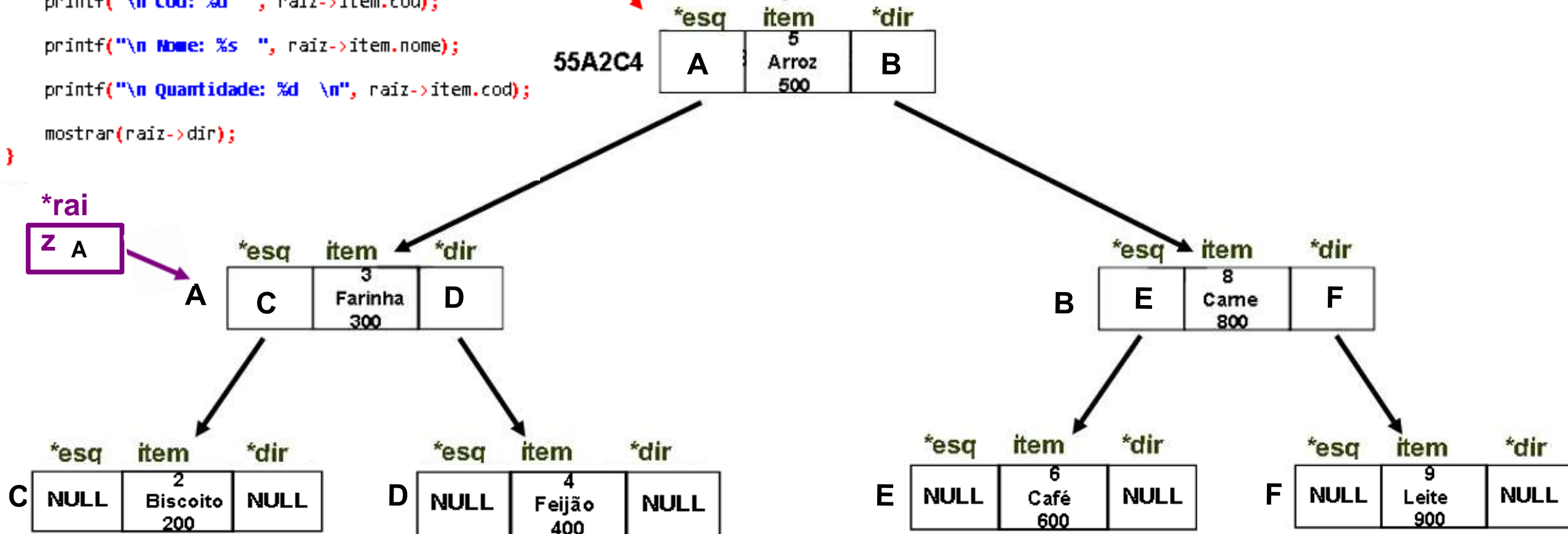
*rai

55A2C4

*raiz

55A2C4

CONTEÚDO DA ÁRVORE :

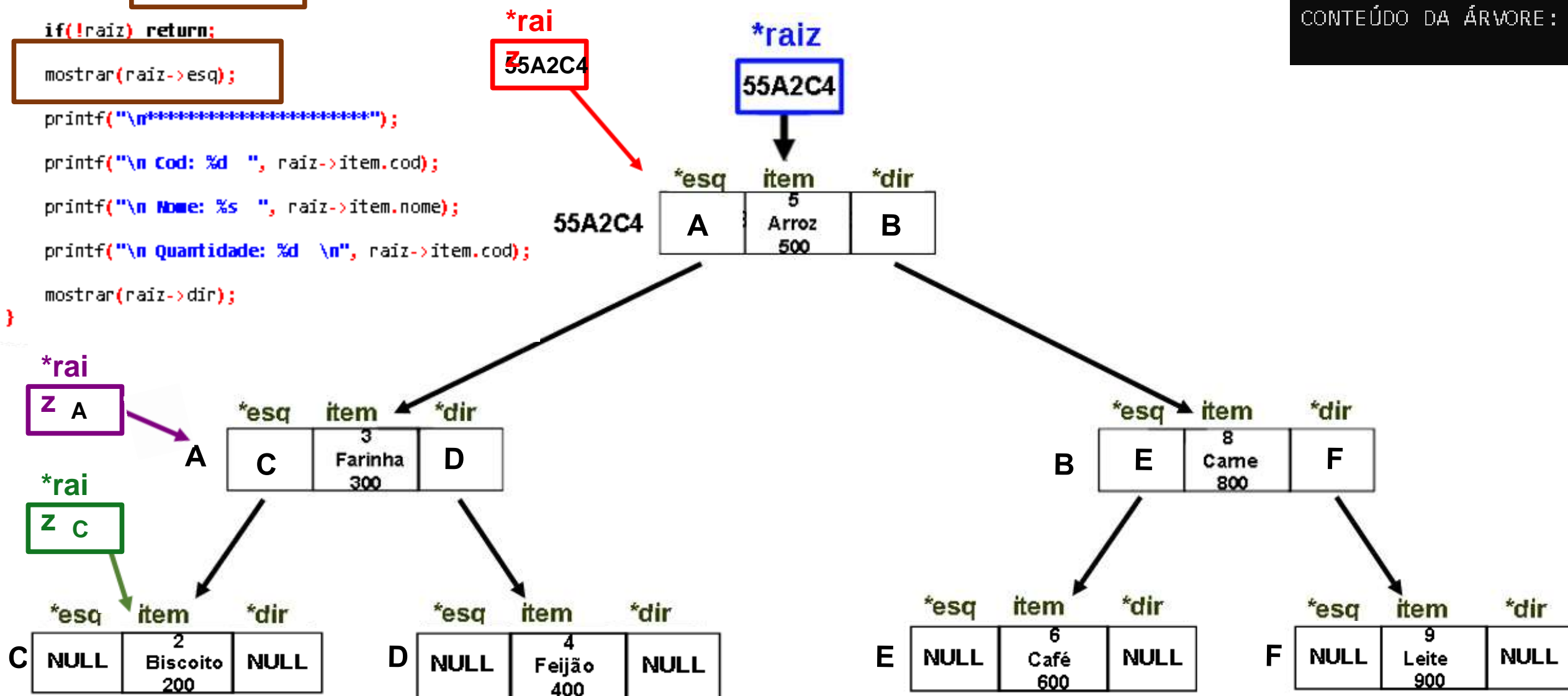



```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);
    printf("\n*****");
    printf("\n Cod: %d ", raiz->item.cod);
    printf("\n Nome: %s ", raiz->item.nome);
    printf("\n Quantidade: %d \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

CONTEÚDO DA ÁRVORE :



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

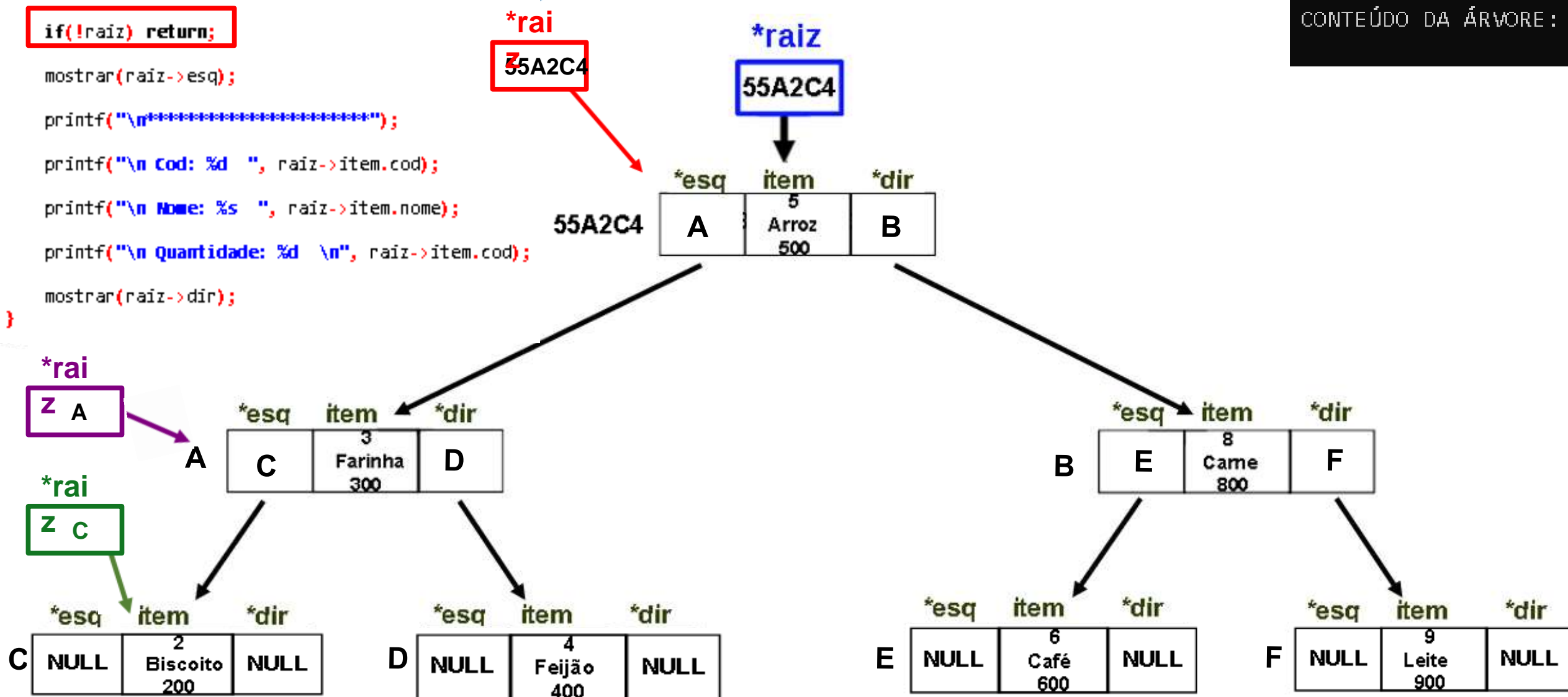
```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

CONTEÚDO DA ÁRVORE :




```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

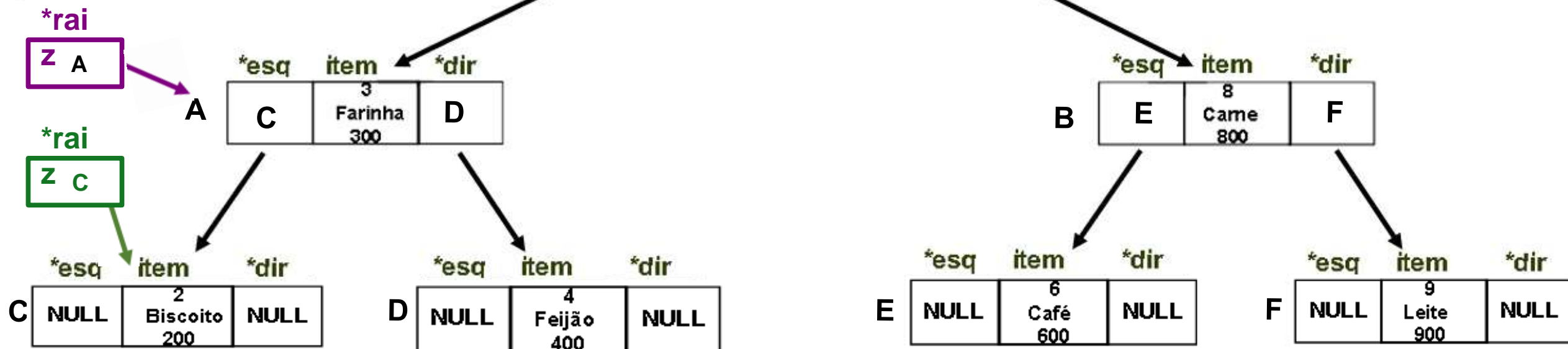
55A2C4

CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

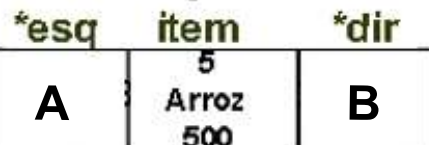
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



***rai**

Z A

A



***rai**

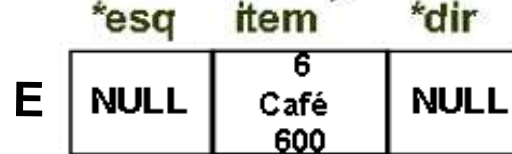
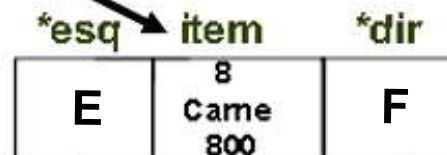
Z C



***rai**

Z NULL

B



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

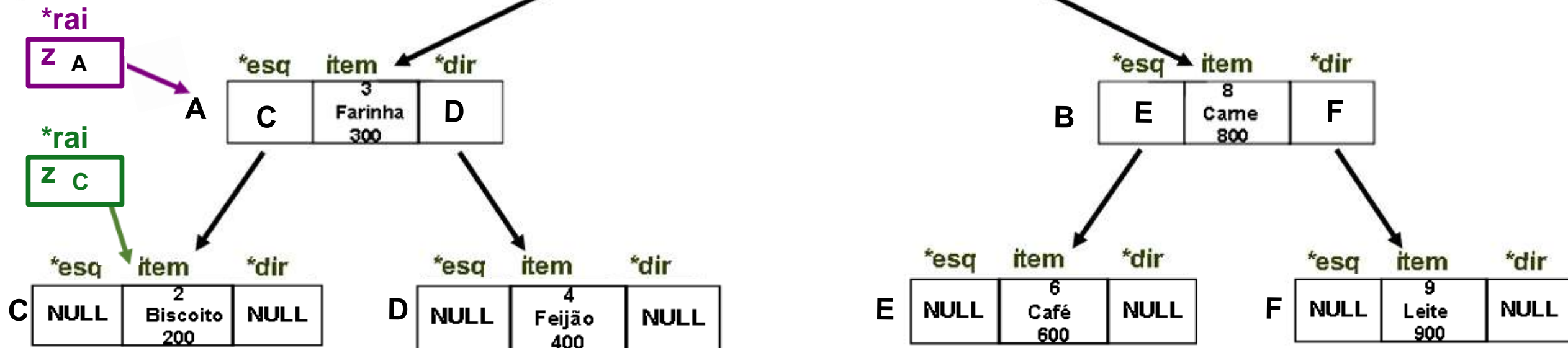
55A2C4

CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200



***rai**

7 NULL

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

```
if(!raiz) return;
```

```
printf("\n*****");
```

```
printf("\n Cod: %d  ", raiz->item.cod);
```

```
printf("\n Nome: %s ", raiz->item.nome);
```

```
printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
mostrar(raiz->dir);
```

***rai**

Z A

***rai**

z c



Cod: 2

Nome: Biscoito

Quantidade: 200

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

*rai

Z A

*rai

55A2C4

*raiz

55A2C4

CONTEÚDO DA ÁRVORE:

Cod: 2

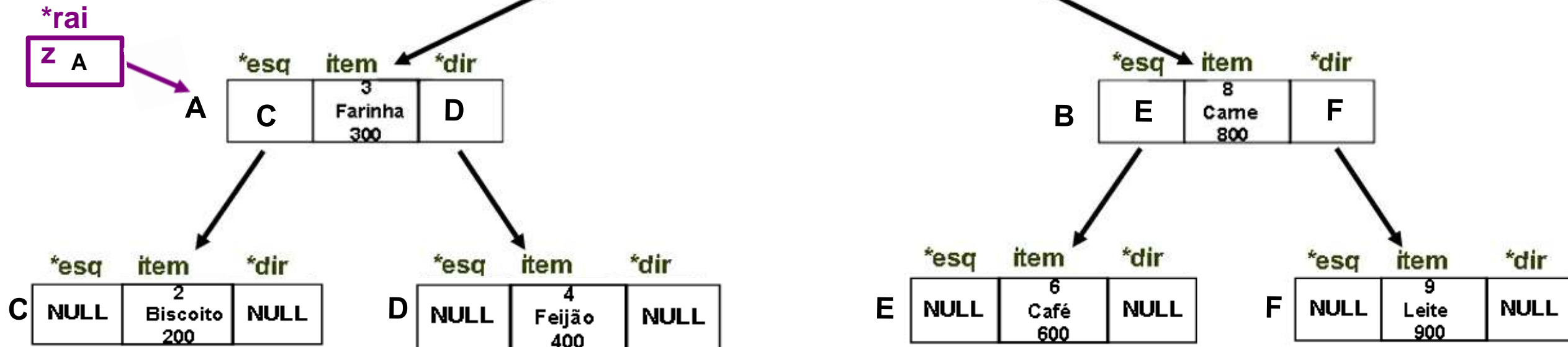
Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300



```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);
    printf("\n*****");
    printf("\n Cod: %d  ", raiz->item.cod);
    printf("\n Nome: %s  ", raiz->item.nome);
    printf("\n Quantidade: %d  \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

***rai**
Z 55A2C4

***raiz**
55A2C4

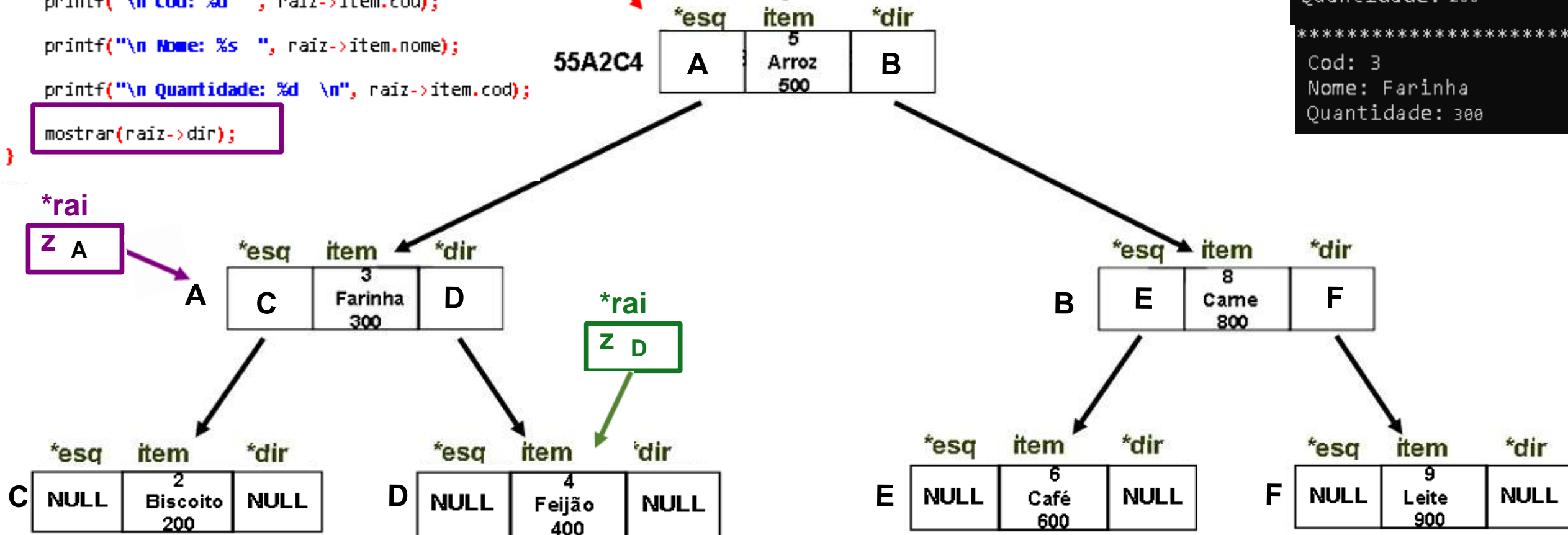
CONTEÚDO DA ÁRVORE:

Cod: 2
Nome: Biscoito
Quantidade: 200

Cod: 3
Nome: Farinha
Quantidade: 300

***rai**
Z A

***rai**
Z D




```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);
    printf("\n*****");
    printf("\n Cod: %d  ", raiz->item.cod);
    printf("\n Nome: %s  ", raiz->item.nome);
    printf("\n Quantidade: %d  \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

***rai**
Z 55A2C4

***raiz**
55A2C4

CONTEÚDO DA ÁRVORE:

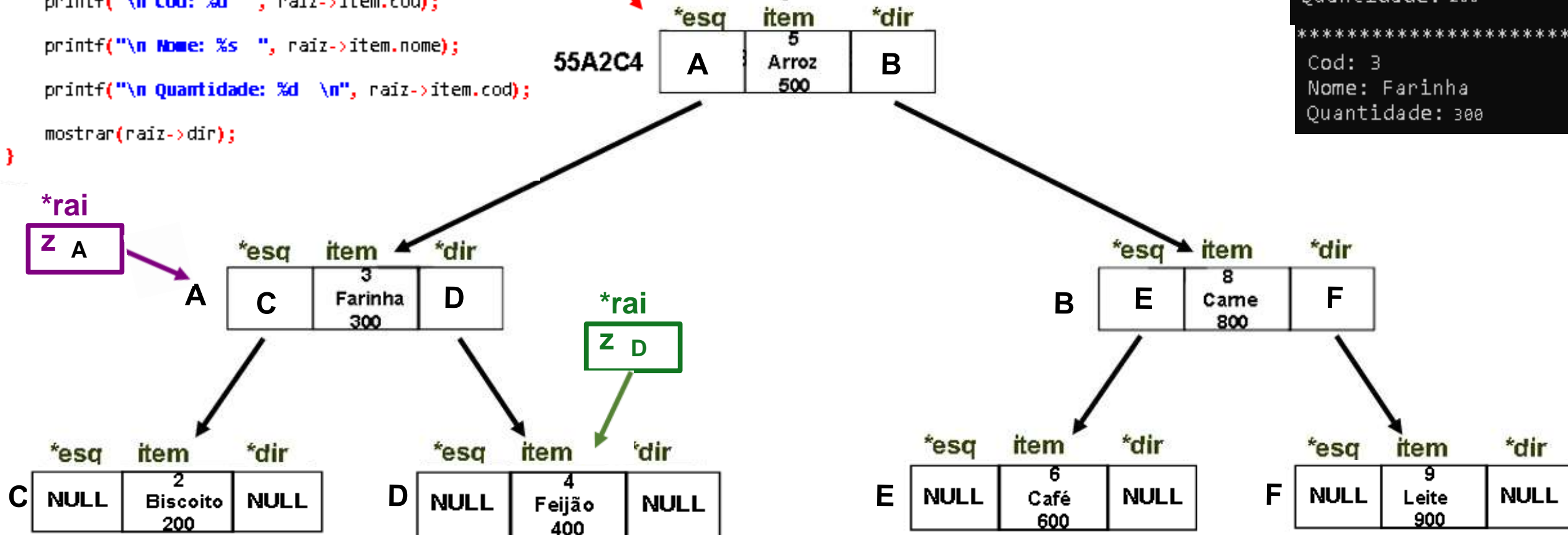
Cod: 2
Nome: Biscoito
Quantidade: 200

Cod: 3
Nome: Farinha
Quantidade: 300

***rai**
Z A

***rai**
Z D

***rai**
Z NULL



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

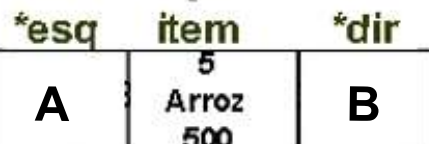
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



***rai**

Z A

A



***rai**

Z D

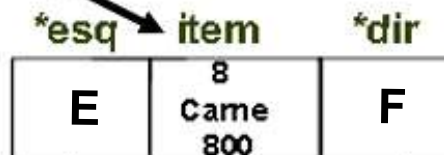
D



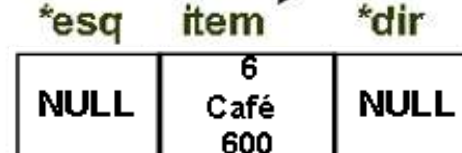
***rai**

7 NULL

B



E



F



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE


```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

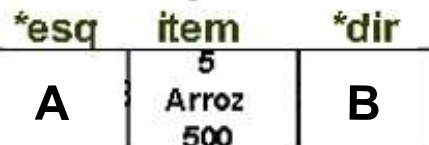
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



***rai**

Z A

A



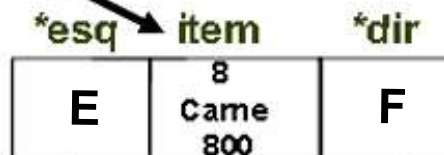
***rai**

Z D

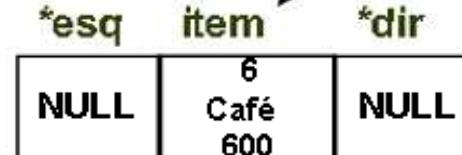
D



B



E



F



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

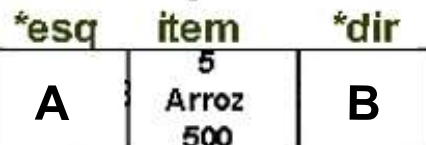
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



***rai**

Z A

A



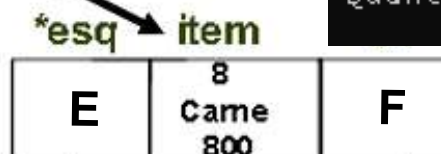
***rai**

Z D

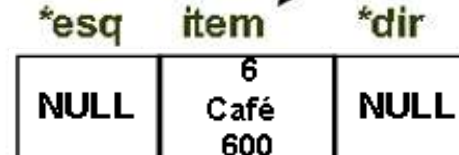
D



B



E



F



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

Cod: 4

Nome: Feijao

Quantidade: 400

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);
    printf("\n*****");
    printf("\n Cod: %d  ", raiz->item.cod);
    printf("\n Nome: %s  ", raiz->item.nome);
    printf("\n Quantidade: %d  \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

***rai**
Z 55A2C4

***raiz**
55A2C4

CONTEÚDO DA ÁRVORE:

Cod: 2
Nome: Biscoito
Quantidade: 200

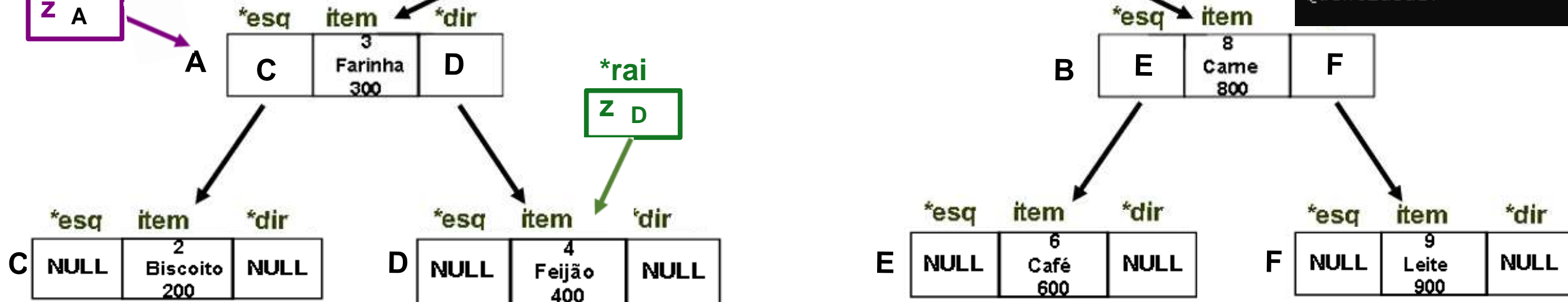
Cod: 3
Nome: Farinha
Quantidade: 300

Cod: 4
Nome: Feijao
Quantidade: 400

***rai**
Z A

***rai**
Z D

***rai**
Z NULL



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

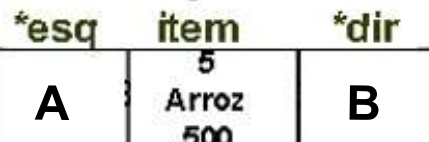
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



***rai**

Z A

A



***rai**

Z D

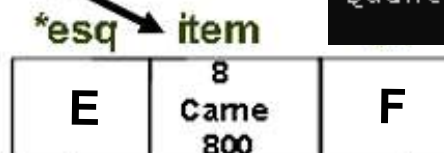
D



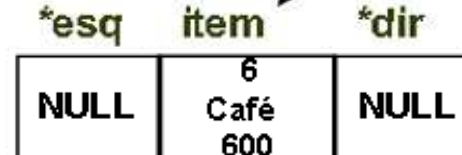
***rai**

7 NULL

B



E



F



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

Cod: 4

Nome: Feijao

Quantidade: 400

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

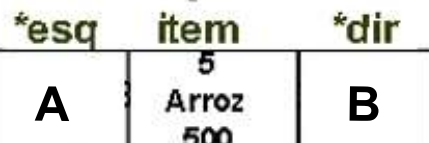
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



***rai**

Z A

A



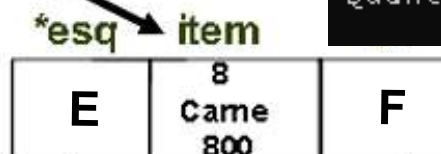
***rai**

Z D

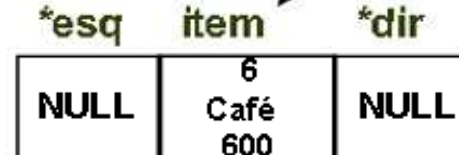
D



B



E



F



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

Cod: 4

Nome: Feijao

Quantidade: 400

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

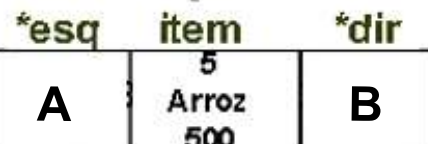
***rai**

55A2C4

***raiz**

55A2C4

55A2C4



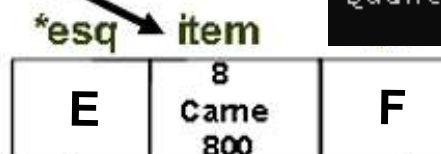
***rai**

Z A

A



B



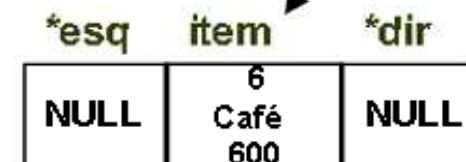
C



D



E



F



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

Cod: 4

Nome: Feijao

Quantidade: 400

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE


```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

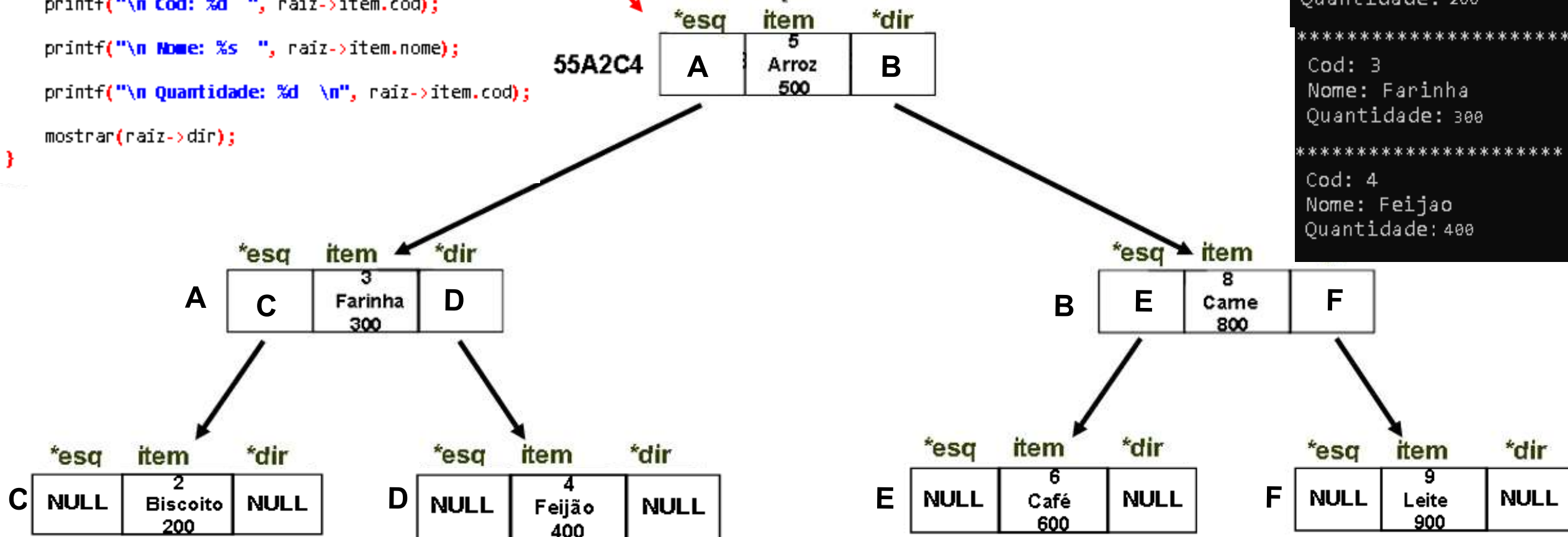
```
}
```

***rai**

55A2C4

***raiz**

55A2C4



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

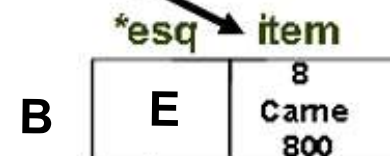
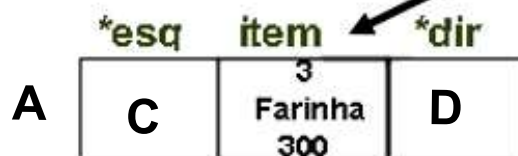
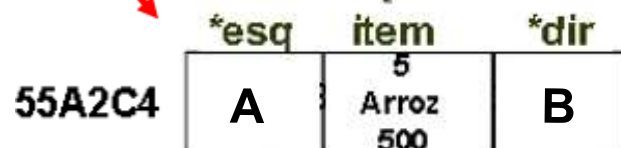
```
}
```

***rai**

55A2C4

***raiz**

55A2C4



CONTEÚDO DA ÁRVORE:

Cod: 2

Nome: Biscoito

Quantidade: 200

Cod: 3

Nome: Farinha

Quantidade: 300

Cod: 4

Nome: Feijao

Quantidade: 400

Cod: 5

Nome: Arroz

Quantidade: 500

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

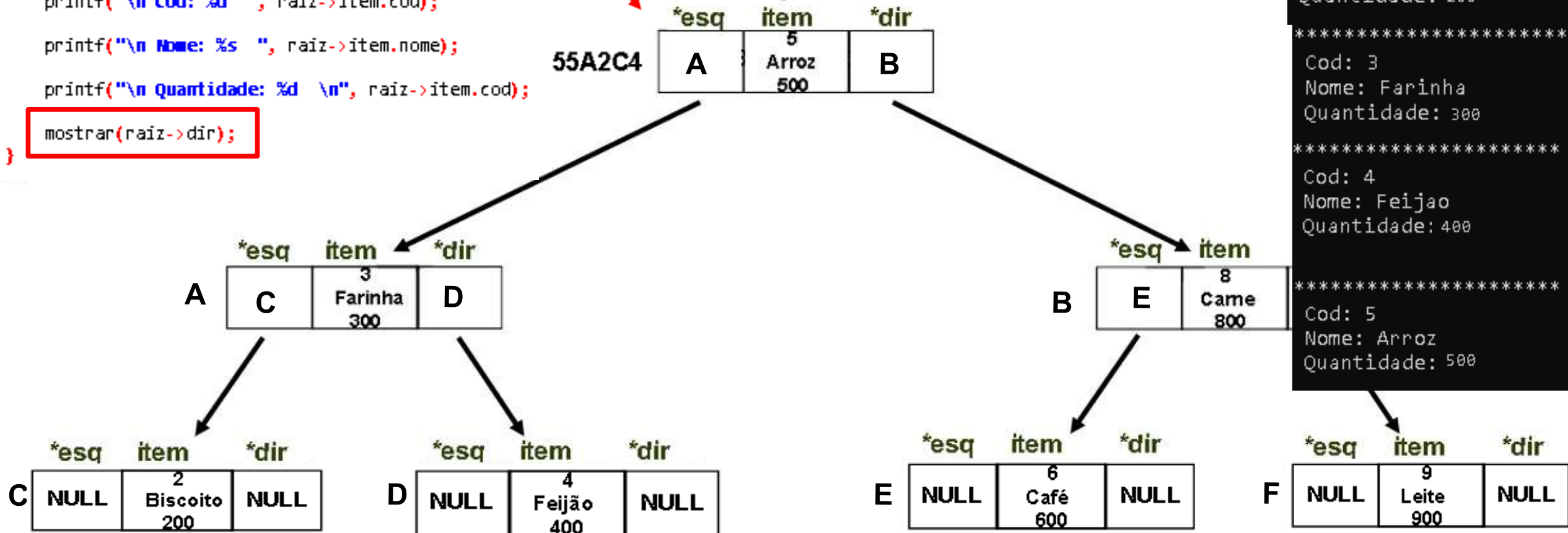

```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);
    printf("\n*****");
    printf("\n Cod: %d  ", raiz->item.cod);
    printf("\n Nome: %s  ", raiz->item.nome);
    printf("\n Quantidade: %d  \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

***rai**
55A2C4

***raiz**
55A2C4



```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);

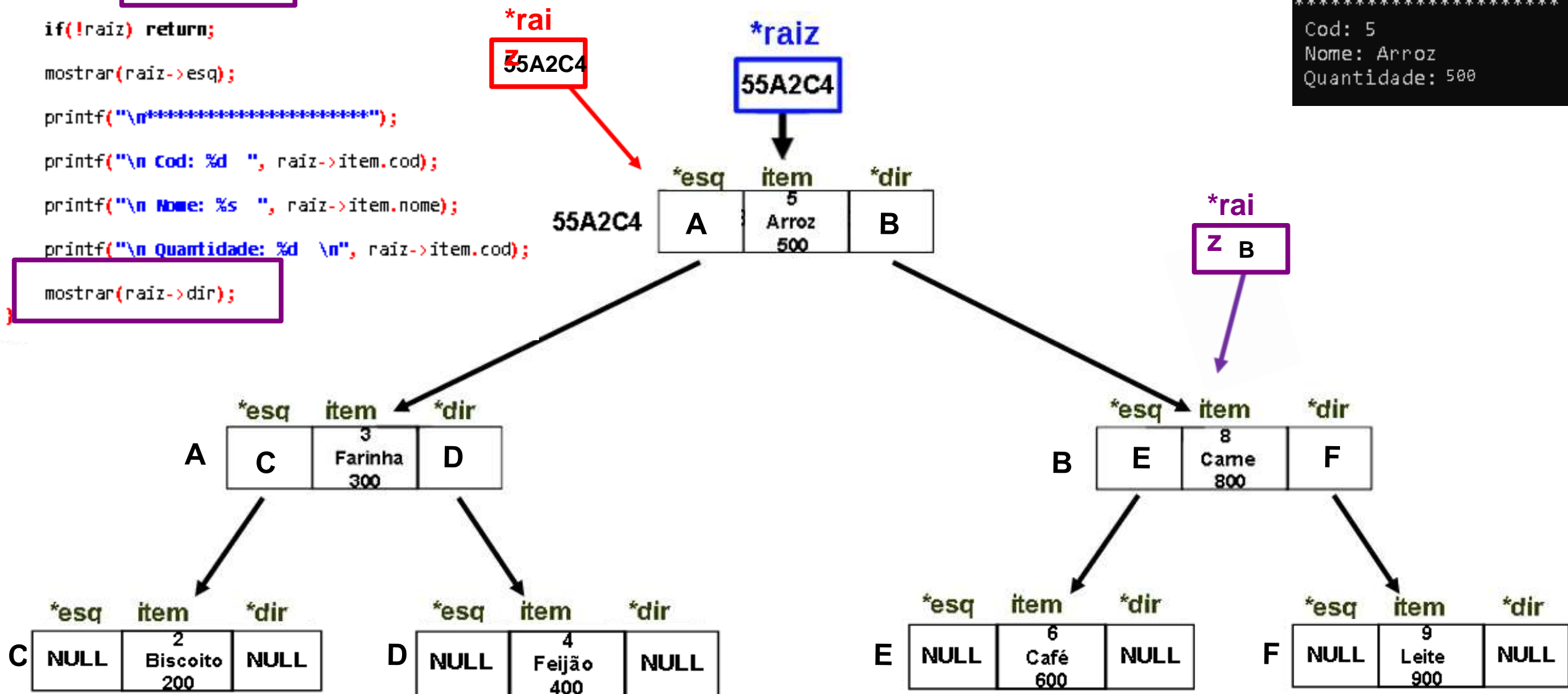
    printf("\n*****");
    printf("\n Cod: %d ", raiz->item.cod);
    printf("\n Nome: %s ", raiz->item.nome);
    printf("\n Quantidade: %d \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

```

*****
Cod: 5
Nome: Arroz
Quantidade: 500

```



```

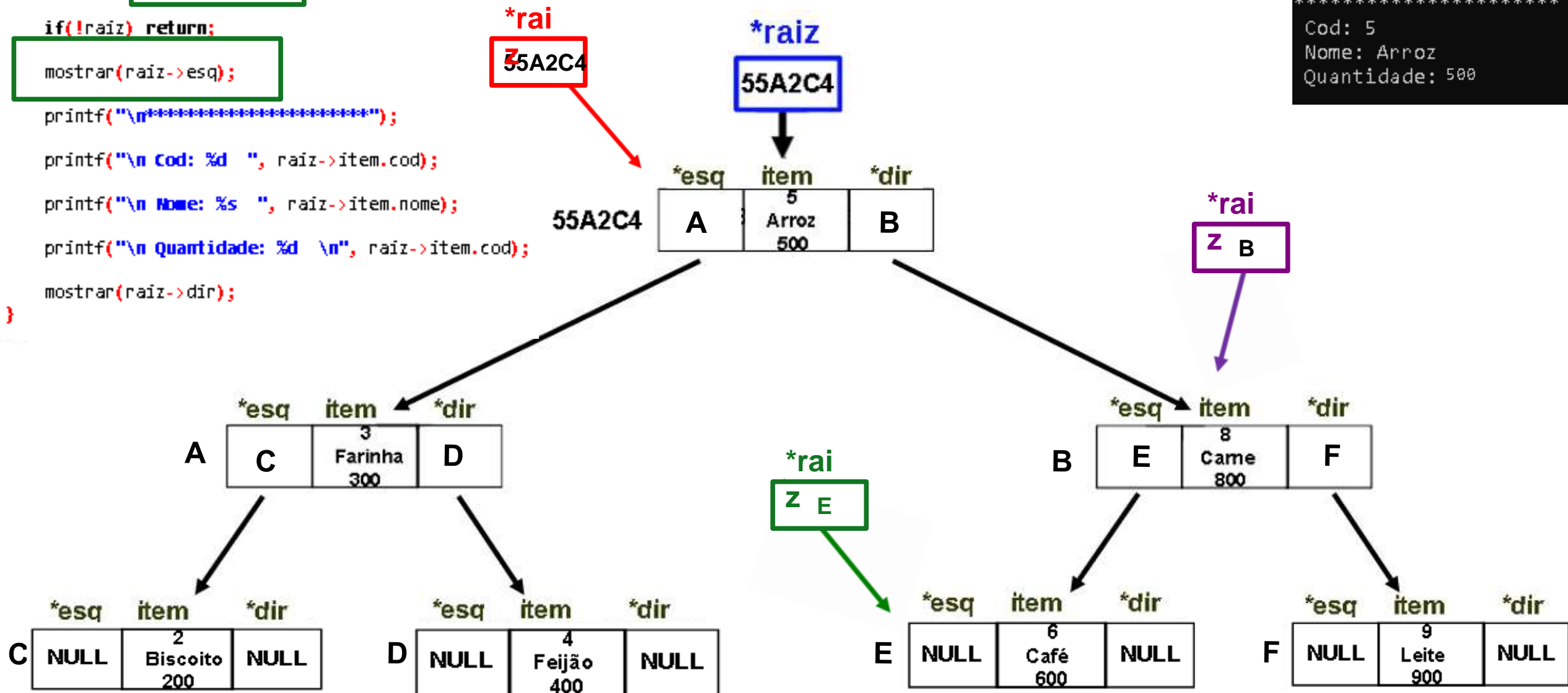
void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);
    printf("\n*****");
    printf("\n Cod: %d ", raiz->item.cod);
    printf("\n Nome: %s ", raiz->item.nome);
    printf("\n Quantidade: %d \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

```

*****
Cod: 5
Nome: Arroz
Quantidade: 500

```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

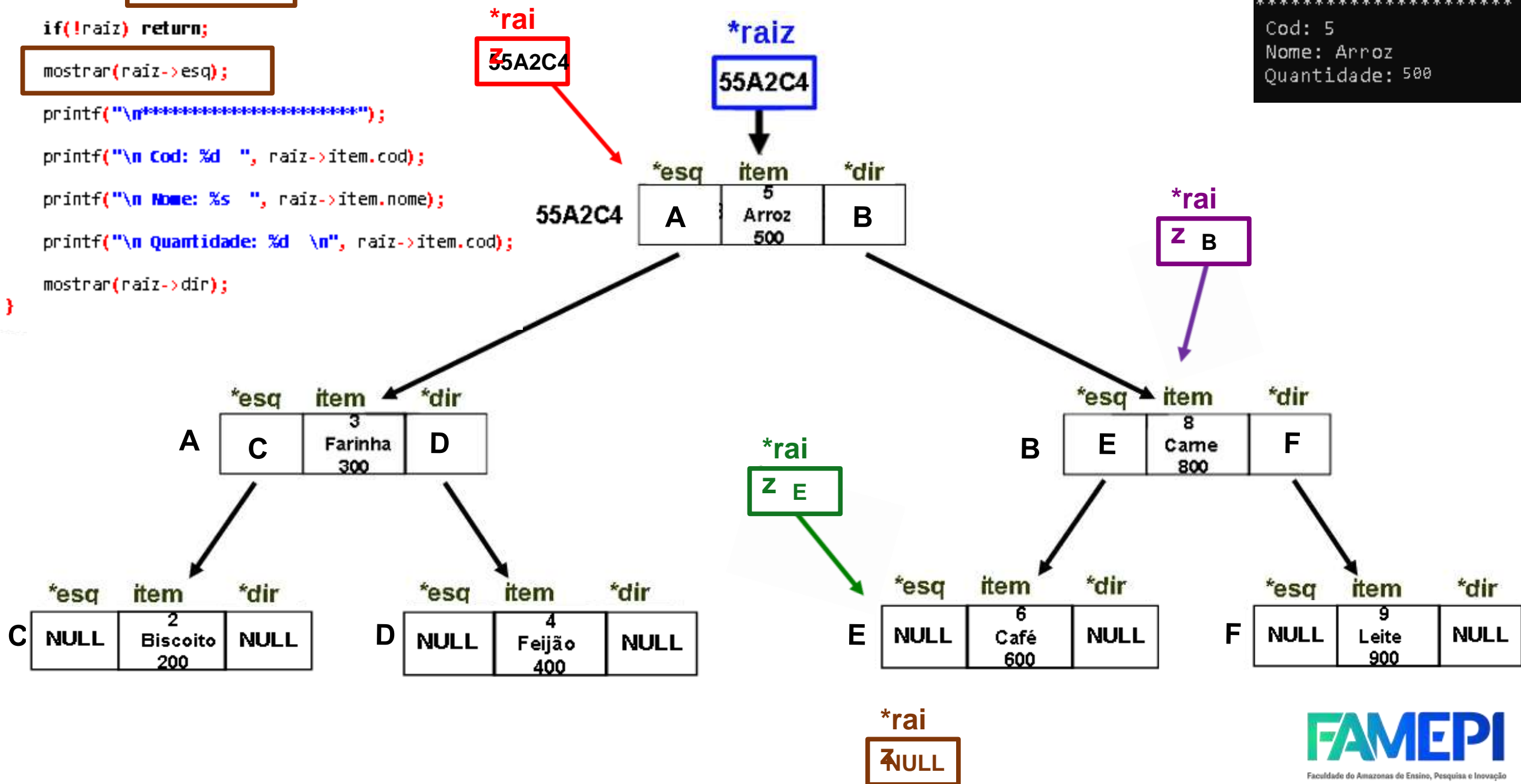
```
    mostrar(raiz->dir);
```

```
}
```

Cod: 5

Nome: Arroz

Quantidade: 500



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

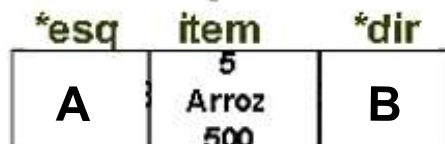
55A2C4

Cod: 5

Nome: Arroz

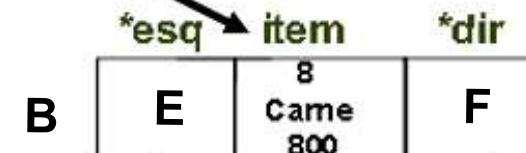
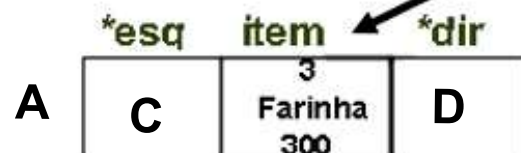
Quantidade: 500

55A2C4



***rai**

Z B



***rai**

Z E



***rai**

Z NULL

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

Cod: 5

Nome: Arroz

Quantidade: 500

***rai**

Z B

***rai**

Z E

C

A

B

F

*esq	item	*dir
C	3 Farinha 300	D

*esq	item	*dir
NULL	2 Biscoito 200	NULL

*esq	item	*dir
NULL	4 Feijão 400	NULL

*esq	item	*dir
NULL	6 Café 600	NULL

*esq	item	*dir
NULL	9 Leite 900	NULL

FAMEPI

Faculdade do Amazonas do Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE


```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

55A2C4

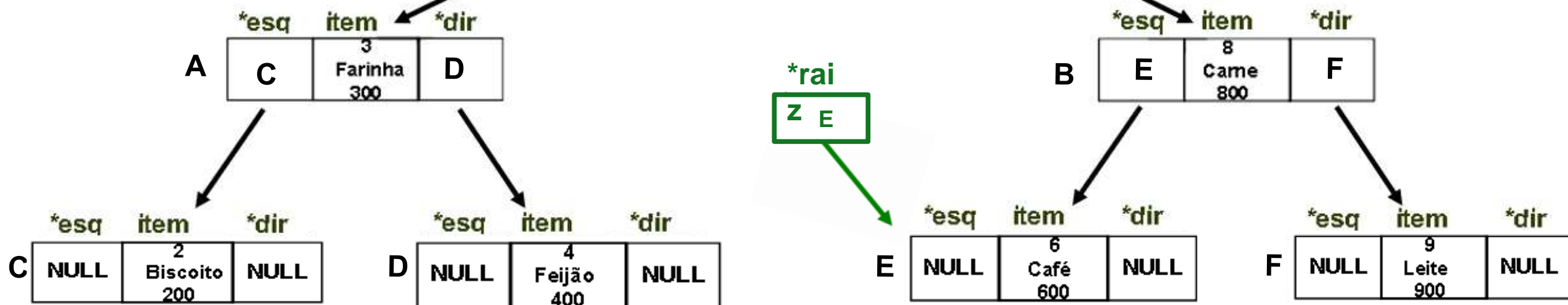
***rai**

Z B

***rai**

Z E

```
*****
Cod: 5
Nome: Arroz
Quantidade: 500
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
```



```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);

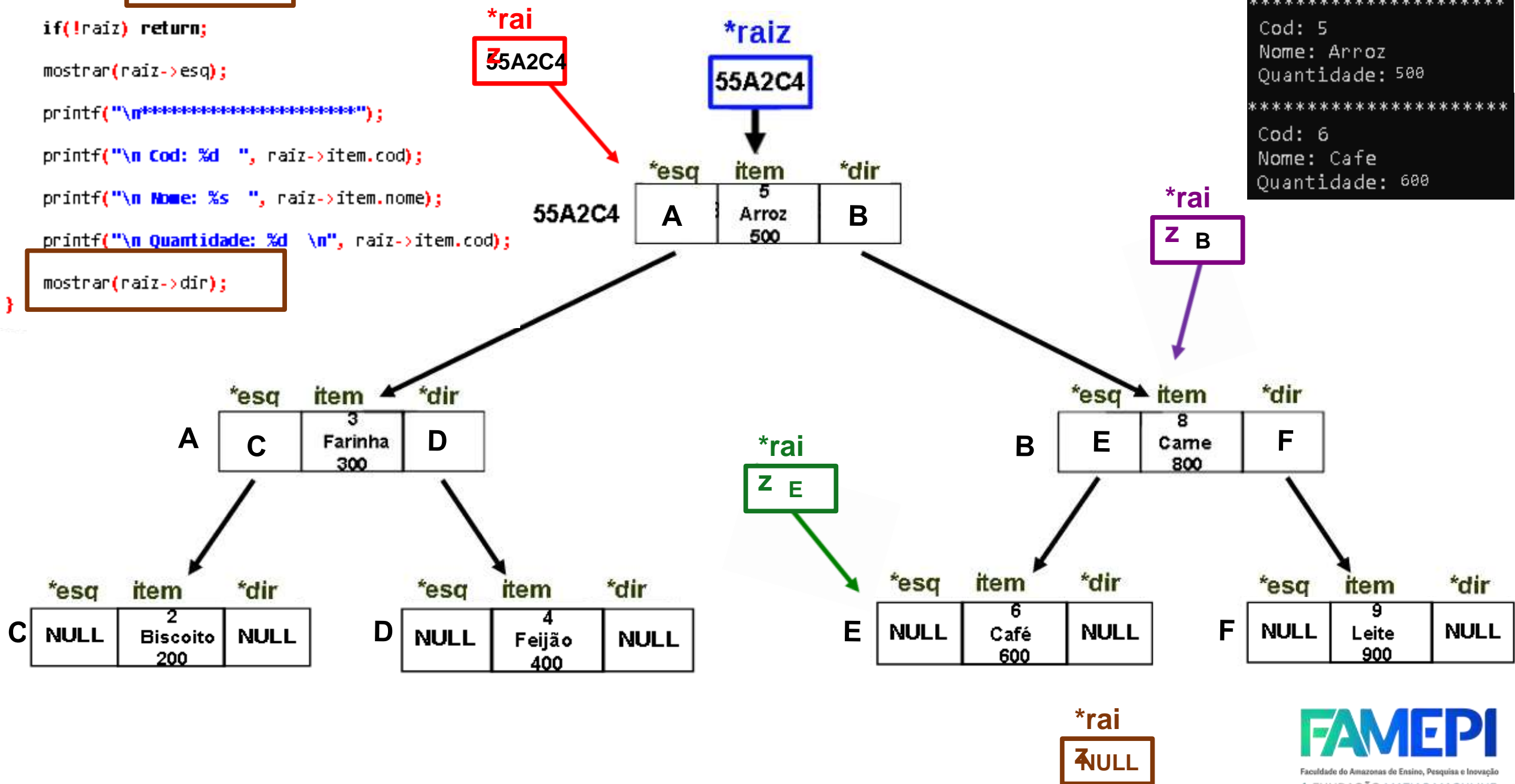
    printf("\n*****");
    printf("\n Cod: %d  ", raiz->item.cod);
    printf("\n Nome: %s  ", raiz->item.nome);
    printf("\n Quantidade: %d  \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

```

*****
Cod: 5
Nome: Arroz
Quantidade: 500
*****
Cod: 6
Nome: Cafe
Quantidade: 600

```




```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

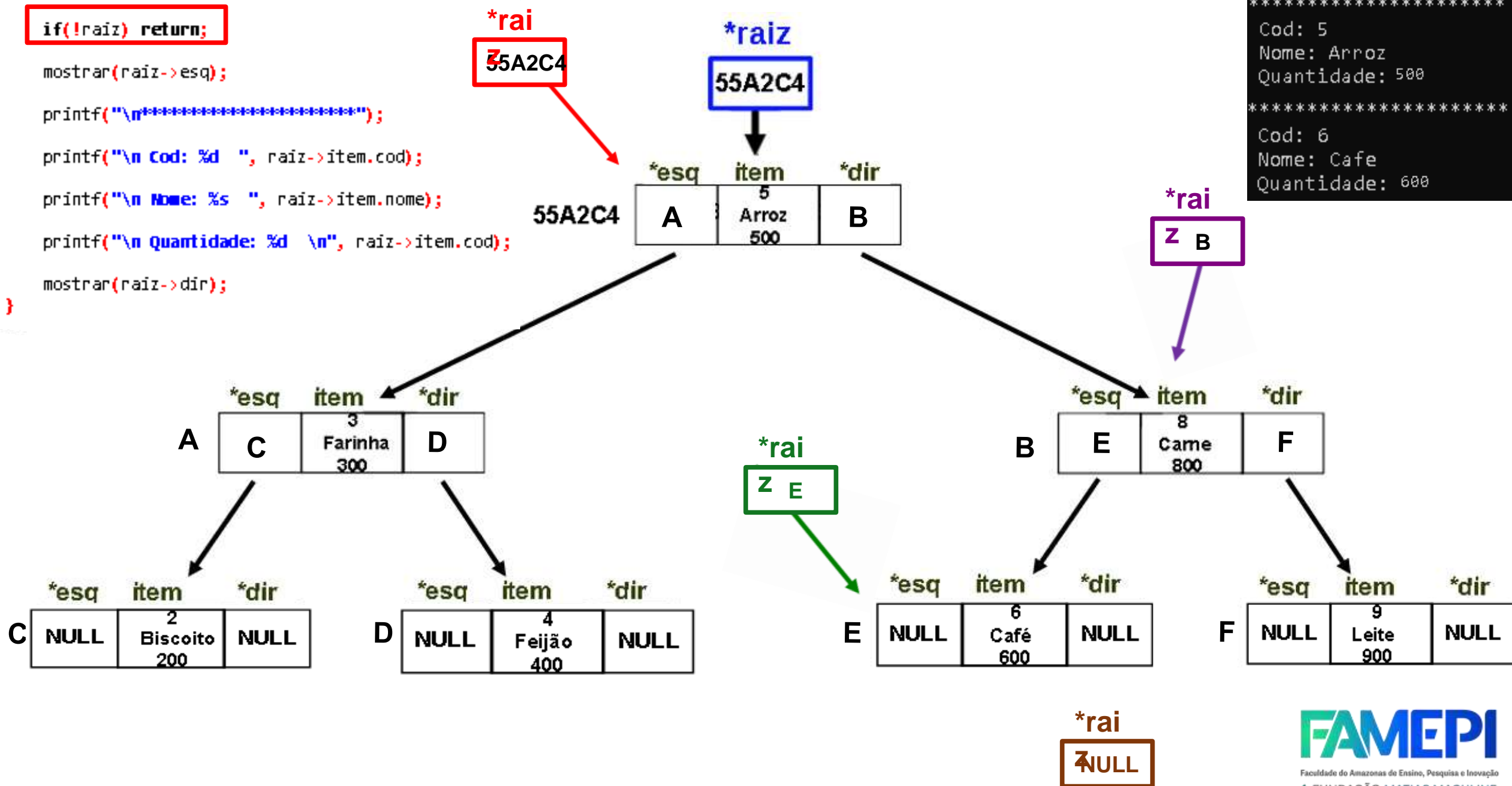
```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

55A2C4

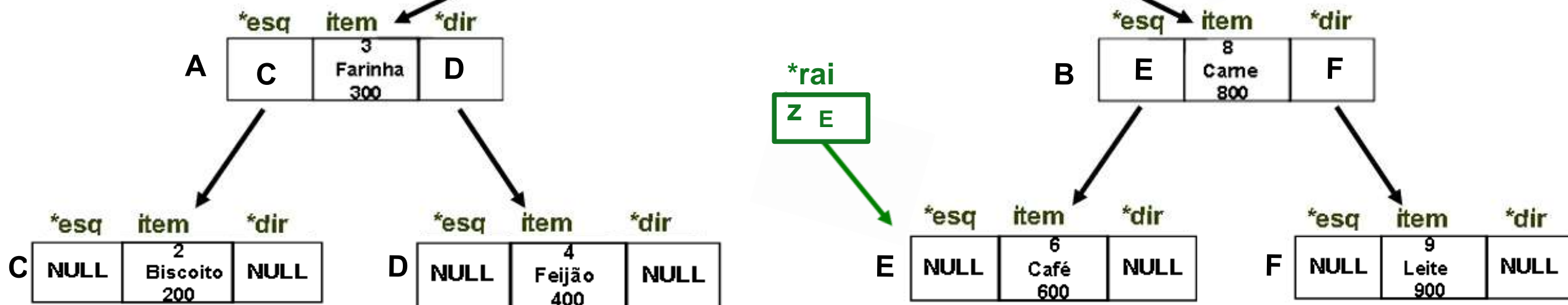
***rai**

Z B

***rai**

Z E

```
*****
Cod: 5
Nome: Arroz
Quantidade: 500
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

55A2C4

***rai**

Z B

Cod: 5

Nome: Arroz

Quantidade: 500

Cod: 6

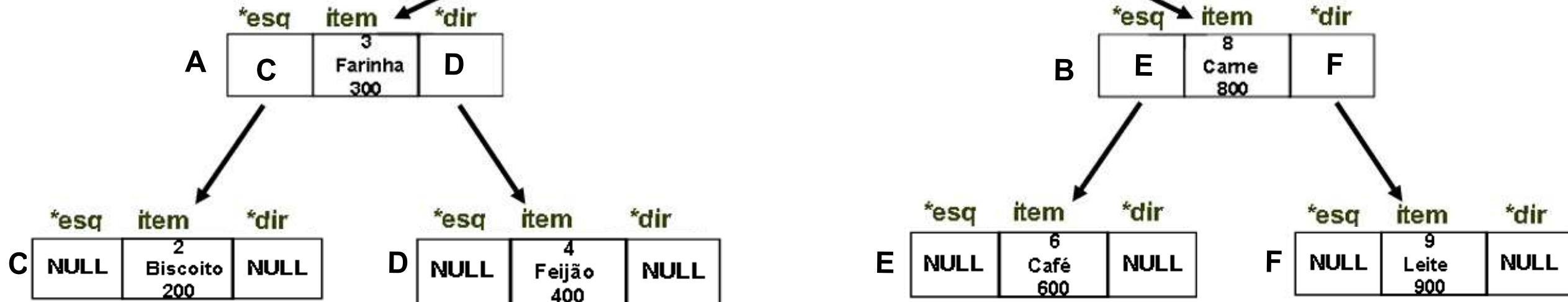
Nome: Cafe

Quantidade: 600

Cod: 8

Nome: Carne

Quantidade: 800



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

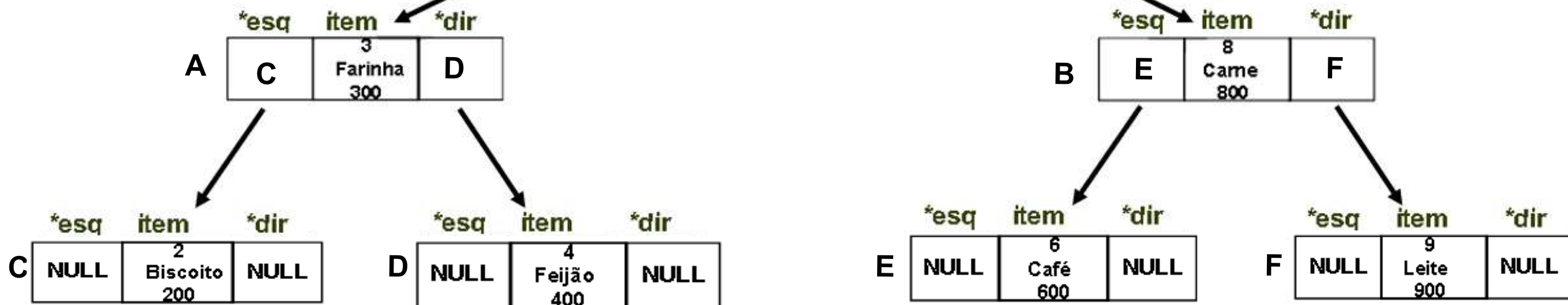
55A2C4

55A2C4

***rai**

Z B

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
```



```

void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);

    printf("\n*****");
    printf("\n Cod: %d ", raiz->item.cod);
    printf("\n Nome: %s ", raiz->item.nome);
    printf("\n Quantidade: %d \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

***rai**
Z 55A2C4

***raiz**
55A2C4

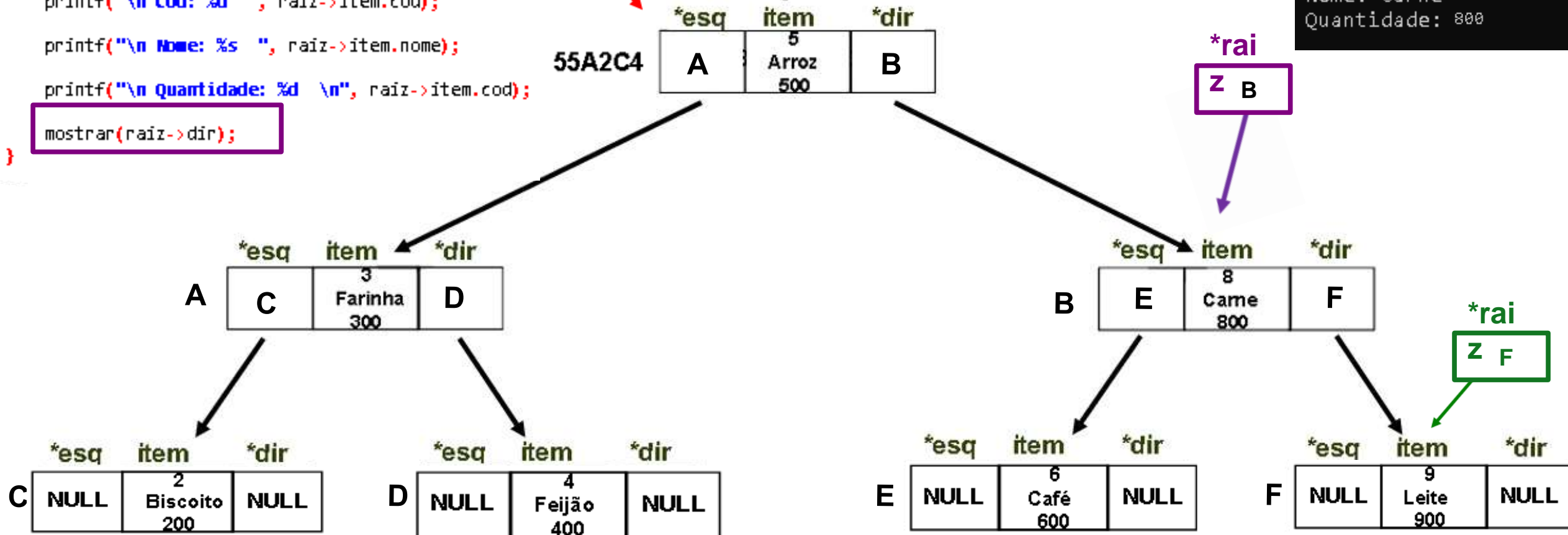
```

*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800

```

***rai**
Z B

***rai**
Z F



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

55A2C4

***rai**

Z B

***rai**

Z F

C

A

B

E

F

*esq	item	*dir
C	3 Farinha 300	D

*esq	item	*dir
NULL	2 Biscoito 200	NULL

*esq	item	*dir
NULL	4 Feijão 400	NULL

*esq	item	*dir
A	5 Arroz 500	B

*esq	item	*dir
NULL	6 Café 600	NULL

*esq	item	*dir
E	8 Carne 800	F

*esq	item	*dir
NULL	9 Leite 900	NULL

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
```

***rai**

7 NULL


```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

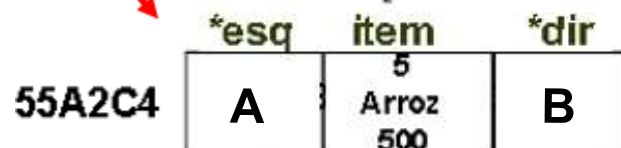
```
}
```

***rai**

55A2C4

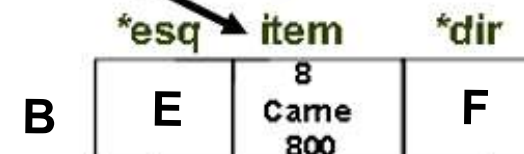
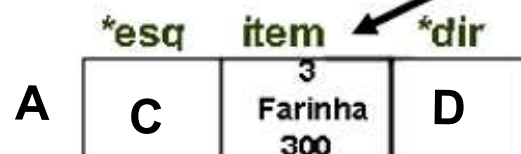
***raiz**

55A2C4



***rai**

Z B



***rai**

Z F



```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
```

***rai**

NULL

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

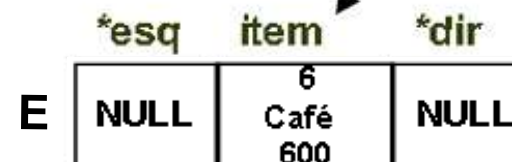
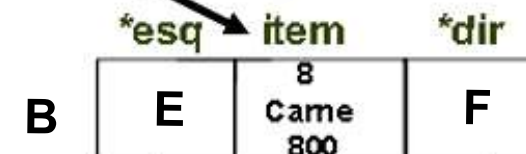
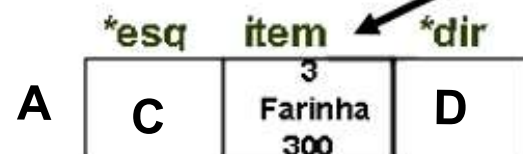
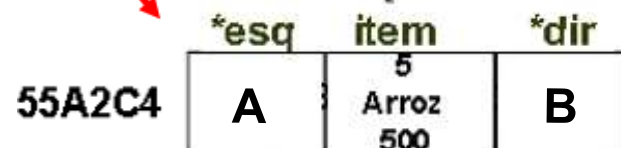
```
}
```

***rai**

55A2C4

***raiz**

55A2C4



***rai**

Z B

***rai**

Z F

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

55A2C4

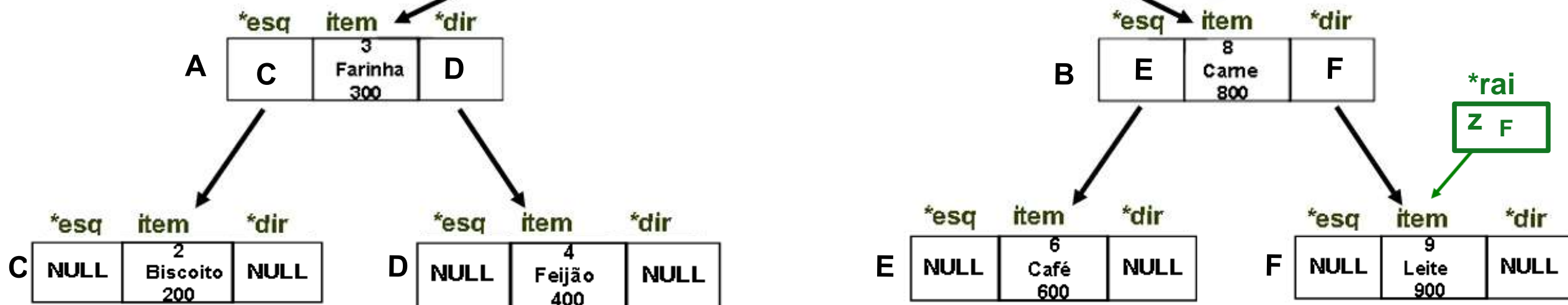
***rai**

Z B

***rai**

Z F

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
Cod: 9
Nome: Leite
Quantidade: 900
*****
```



```

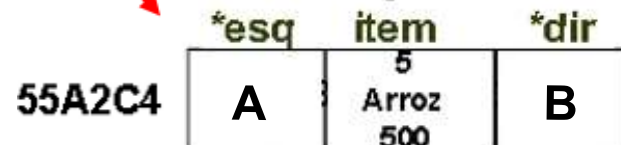
void mostrar(tipo_no *raiz){
    if(!raiz) return;
    mostrar(raiz->esq);

    printf("\n*****");
    printf("\n Cod: %d ", raiz->item.cod);
    printf("\n Nome: %s ", raiz->item.nome);
    printf("\n Quantidade: %d \n", raiz->item.cod);
    mostrar(raiz->dir);
}

```

***rai**
55A2C4

***raiz**
55A2C4

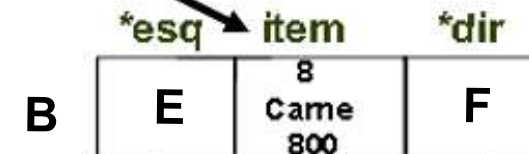
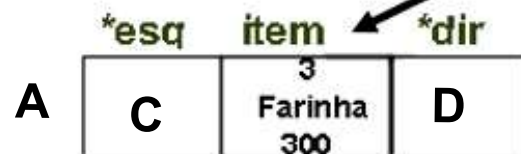


***rai**
Z B

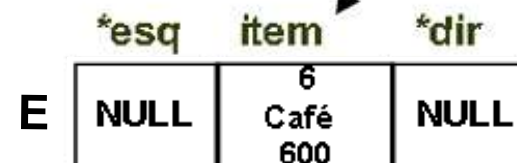
```

*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
Cod: 9
Nome: Leite
Quantidade: 900

```



***rai**
Z F



***rai**
7 NULL

```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***raiz**

55A2C4

***raiz**

55A2C4

55A2C4

***rai**

Z B

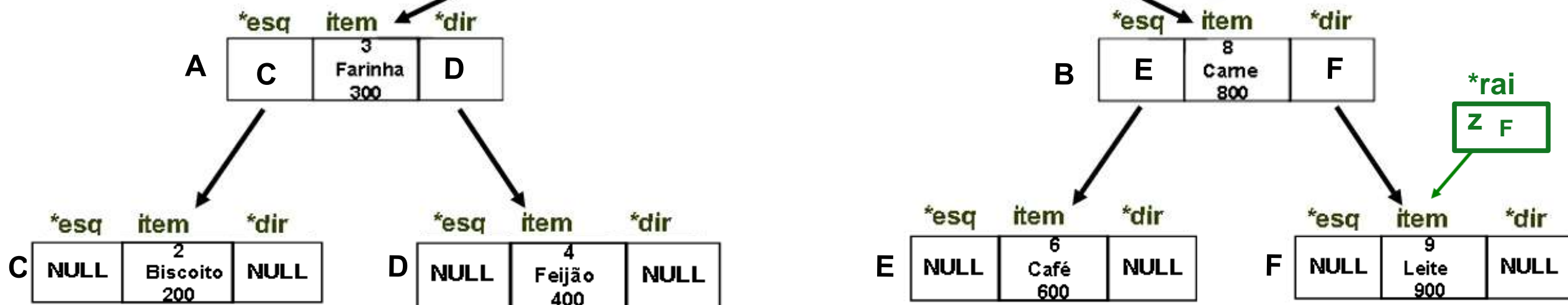
***rai**

Z F

***rai**

7 NULL

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
Cod: 9
Nome: Leite
Quantidade: 900
*****
```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

55A2C4

55A2C4

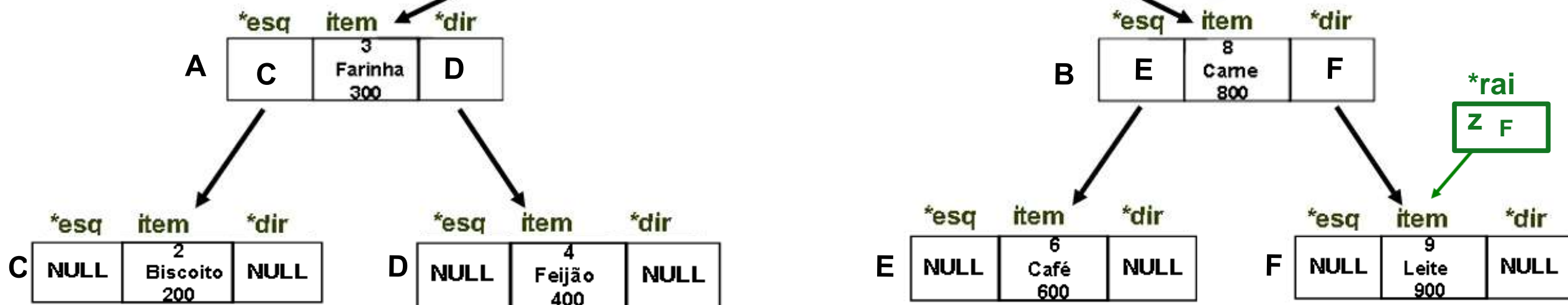
***rai**

Z B

***rai**

Z F

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
Cod: 9
Nome: Leite
Quantidade: 900
*****
```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d ", raiz->item.cod);
```

```
    printf("\n Nome: %s ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

```
}
```

***rai**

55A2C4

***raiz**

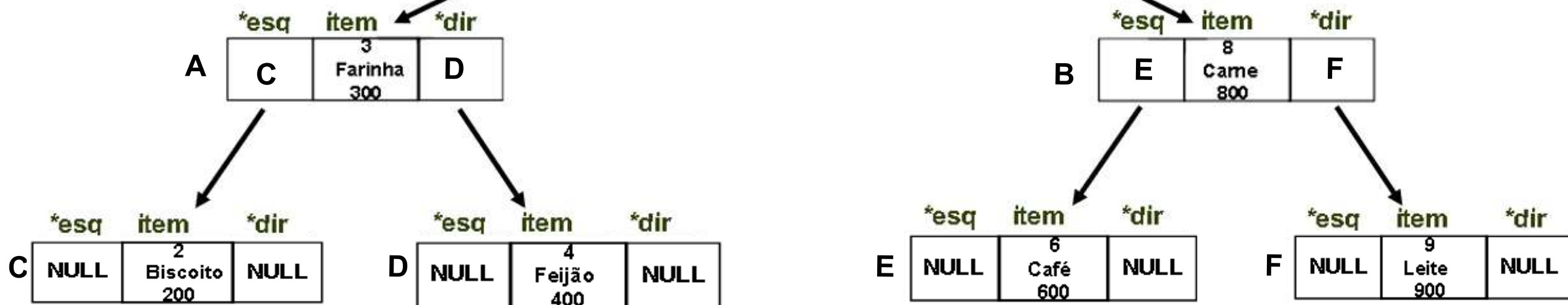
55A2C4

55A2C4

***rai**

Z B

```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
Cod: 9
Nome: Leite
Quantidade: 900
*****
```



```
void mostrar(tipo_no *raiz){
```

```
    if(!raiz) return;
```

```
    mostrar(raiz->esq);
```

```
    printf("\n*****");
```

```
    printf("\n Cod: %d  ", raiz->item.cod);
```

```
    printf("\n Nome: %s  ", raiz->item.nome);
```

```
    printf("\n Quantidade: %d  \n", raiz->item.cod);
```

```
    mostrar(raiz->dir);
```

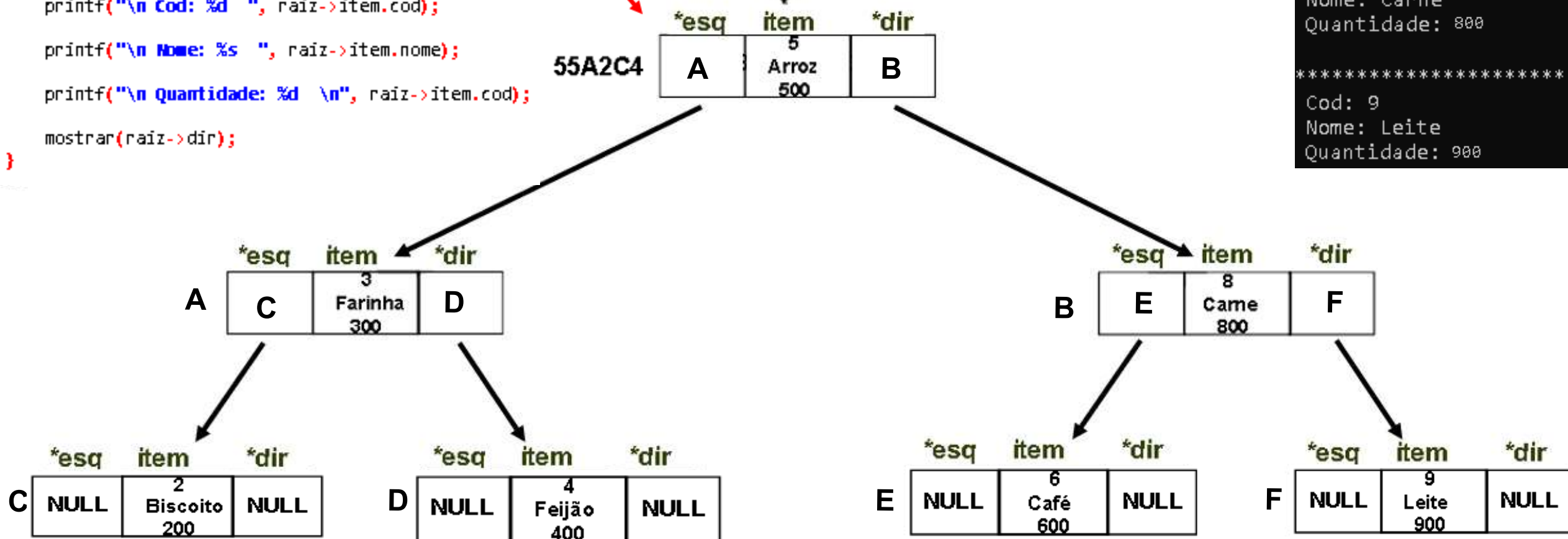
```
}
```

***rai**

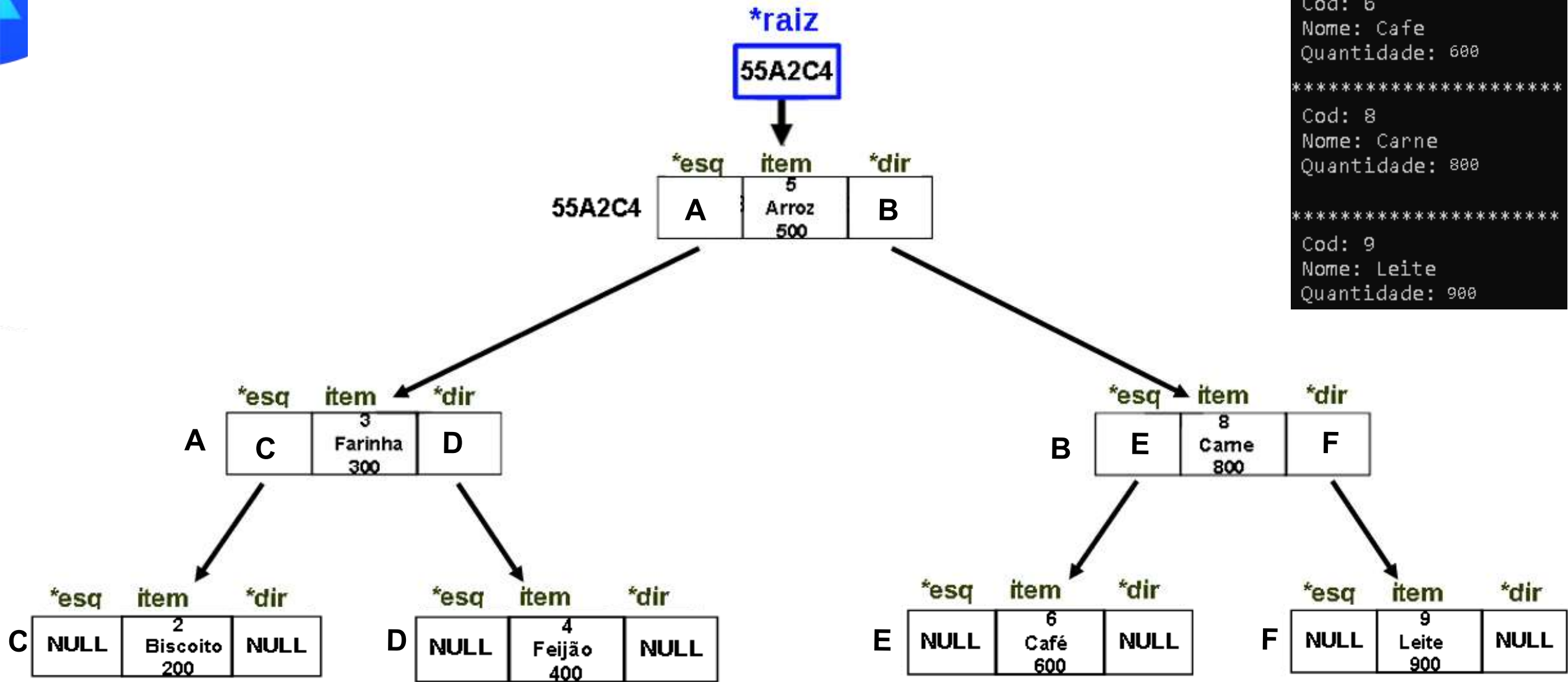
55A2C4

***raiz**

55A2C4



```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****
Cod: 8
Nome: Carne
Quantidade: 800
*****
Cod: 9
Nome: Leite
Quantidade: 900
*****
```

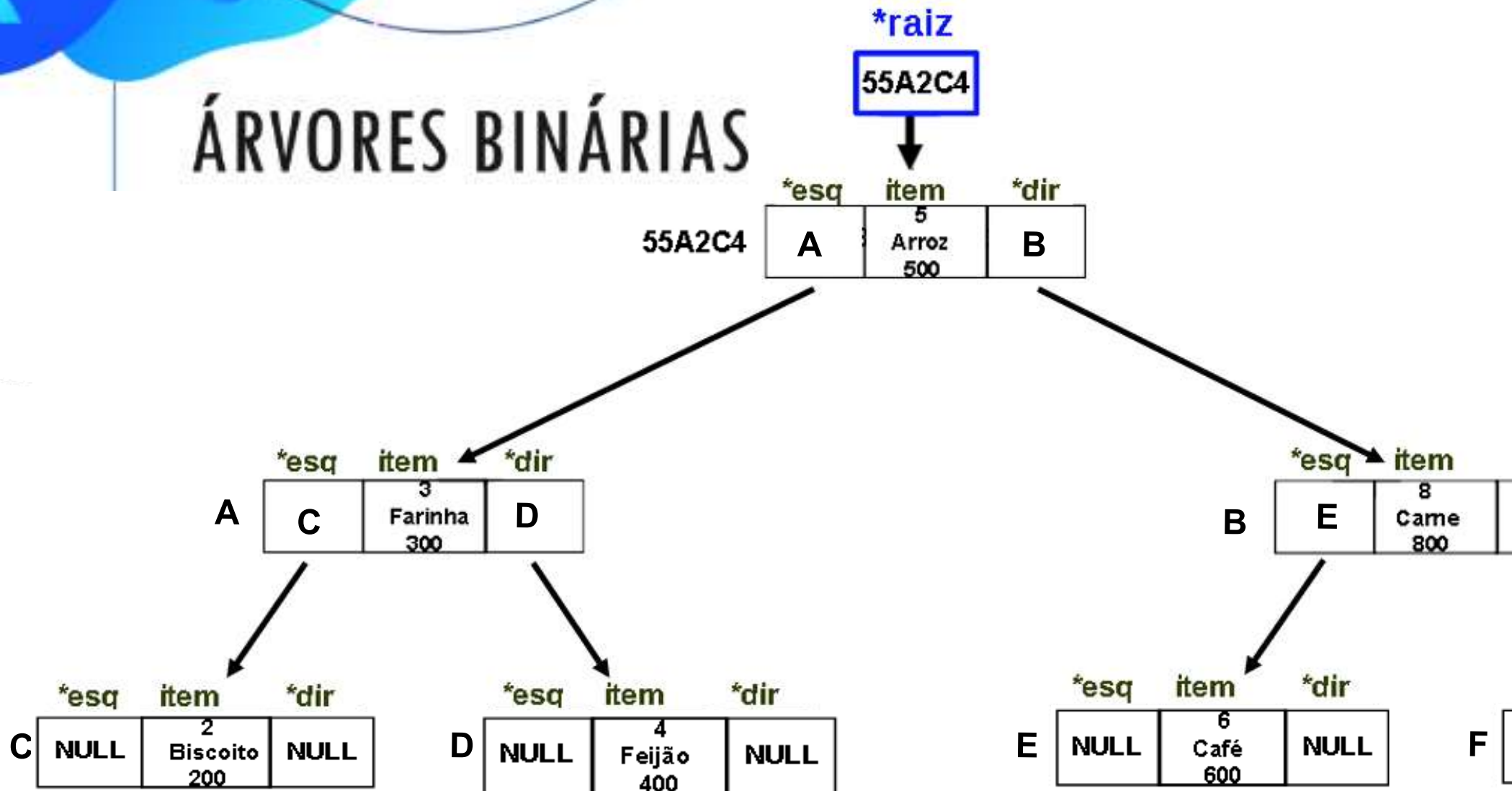



```
*****
Cod: 6
Nome: Cafe
Quantidade: 600
*****

Cod: 8
Nome: Carne
Quantidade: 800
*****

Cod: 9
Nome: Leite
Quantidade: 900
*****
```

ÁRVORES BINÁRIAS



```
CONTEÚDO DA ÁRVORE:
*****
Cod: 2
Nome: Biscoito
Quantidade: 200

*****
Cod: 3
Nome: Farinha
Quantidade: 300

*****
Cod: 4
Nome: Feijao
Quantidade: 400

*****
Cod: 5
Nome: Arroz
Quantidade: 500

*****
Cod: 6
Nome: Cafe
Quantidade: 600

*****
Cod: 8
Nome: Carne
Quantidade: 800

*****
Cod: 9
Nome: Leite
Quantidade: 900
```




ÁRVORES BINÁRIAS

IMPLEMENTAÇÃO

PESQUISAR NA ÁRVORE

ÁRVORES BINÁRIAS

Procedimento para pesquisar na árvore

- Para encontrar um registro com uma chave x :
 - Compare-a com a chave que está na raiz.
 - Se x é menor, vá para a subárvore esquerda.
 - Se x é maior, vá para a subárvore direita.
 - Repita o processo recursivamente, até que a chave procurada seja encontrada ou um nó folha seja atingido.

ÁRVORES BINÁRIAS

```
int main( ) {  
    setlocale(LC_ALL, "");  
  
    int op, cod;  
  
    tipo_no *raiz = NULL;  
  
    tipo_item aux;
```

op

3

cod

*rai

55A2C4

```
do{  
    system("cls");  
  
    op = menu();  
  
    switch(op){  
  
        case 0: printf("\n\n\n\t Fim do Programa");  
                return(0);  
  
        case 1: aux = preencher_item();  
                if(!raiz) raiz = inserir(aux, raiz, raiz);  
                else inserir(aux, raiz, raiz);  
                break;  
  
        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                mostrar(raiz);  
                getch(); // segura a tela  
                break;  
  
        case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                printf("\n\n Digite o código do item a pesquisar: ");  
                scanf("%d", &cod);  
                pesquisar(cod, raiz);  
                getch(); // segura a tela  
                break;
```

```
***** MENU *****  
[1] - INSERIR  
[2] - MOSTRAR  
[3] - PESQUISAR  
[4] - EXCLUIR  
[0] - SAIR  
  
Digite sua opção: 3
```

ÁRVORES BINÁRIAS

```
void pesquisar(int codPesq, tipo_no *raiz){  
  
    if(!raiz) {  
        printf("\n\n O item de código %d não está na árvore", codPesq);  
        return;  
    }  
  
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }  
  
    if(codPesq > raiz->item.cod){  
        pesquisar(codPesq, raiz->dir);  
    }  
  
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }  
}
```

```
int main( ) {  
    setlocale(LC_ALL, "");  
  
    int op, cod;  
  
    tipo_no *raiz = NULL;  
  
    tipo_item aux;
```

op

3

cod

4

*raiz

55A2C4

```
do{
```

```
    system("PESQUISA NA ÁRVORE
```

```
    op = m Digite o código do item a pesquisar: 4
```

```
    switch(op){
```

```
        case 0: printf("\n\n\n\t Fim do Programa");  
                return(0);
```

```
        case 1: aux = preencher_item();  
                if(!raiz) raiz = inserir(aux, raiz, raiz);  
                else inserir(aux, raiz, raiz);  
                break;
```

```
        case 2: printf("\n\n CONTEÚDO DA ÁRVORE: ");  
                mostrar(raiz);  
                getch(); // segura a tela  
                break;
```

```
        case 3: printf("\n\n PESQUISA NA ÁRVORE ");  
                printf("\n\n Digite o código do item a pesquisar: ");  
                scanf("%d", &cod);  
                pesquisar(cod, raiz);  
                getch(); // segura a tela  
                break;
```

```
***** MENU *****
```

```
[1] - INSERIR  
[2] - MOSTRAR  
[3] - PESQUISAR  
[4] - EXCLUIR  
[0] - SAIR
```

```
Digite sua opção: 3
```

```
void pesquisar(int codPesq, tipo_no *raiz){
```

cod

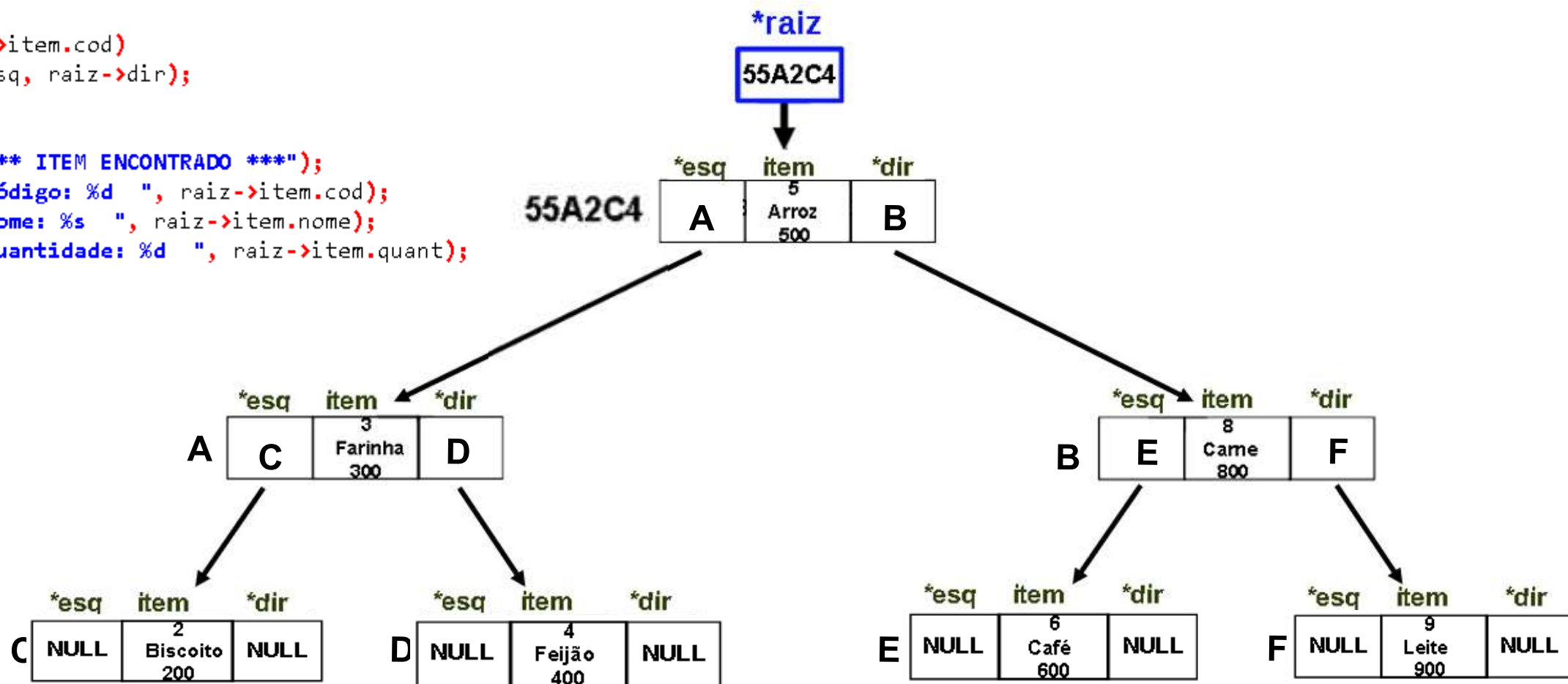
4

```
if(!raiz) {  
    printf("\n\n 0 item de código %d não está na árvore", codPesq);  
    return;  
}
```

```
if(codPesq < raiz->item.cod){  
    pesquisar(codPesq, raiz->esq);  
    return;  
}
```

```
if(codPesq > raiz->item.cod)  
    pesquisar(codPesq, raiz->dir);
```

```
else{  
    printf("\n\n *** ITEM ENCONTRADO ***");  
    printf("\n\n Código: %d ", raiz->item.cod);  
    printf("\n\n Nome: %s ", raiz->item.nome);  
    printf("\n\n Quantidade: %d ", raiz->item.quant);  
}
```



```
void pesquisa(int codPesq, tipo_no *raiz){
```

```
if(!raiz) {  
    printf("\n\n 0 item de código %d não está na árvore", codPesq);  
    return;  
}
```

```
if(codPesq < raiz->item.cod){  
    pesquisa(codPesq, raiz->esq);  
    return;  
}
```

```
if(codPesq > raiz->item.cod)  
    pesquisa(codPesq, raiz->dir);
```

```
else{  
    printf("\n\n *** ITEM ENCONTRADO ***");  
    printf("\n\n Código: %d ", raiz->item.cod);  
    printf("\n\n Nome: %s ", raiz->item.nome);  
    printf("\n\n Quantidade: %d ", raiz->item.quant);  
}
```

cod

4

codpesq

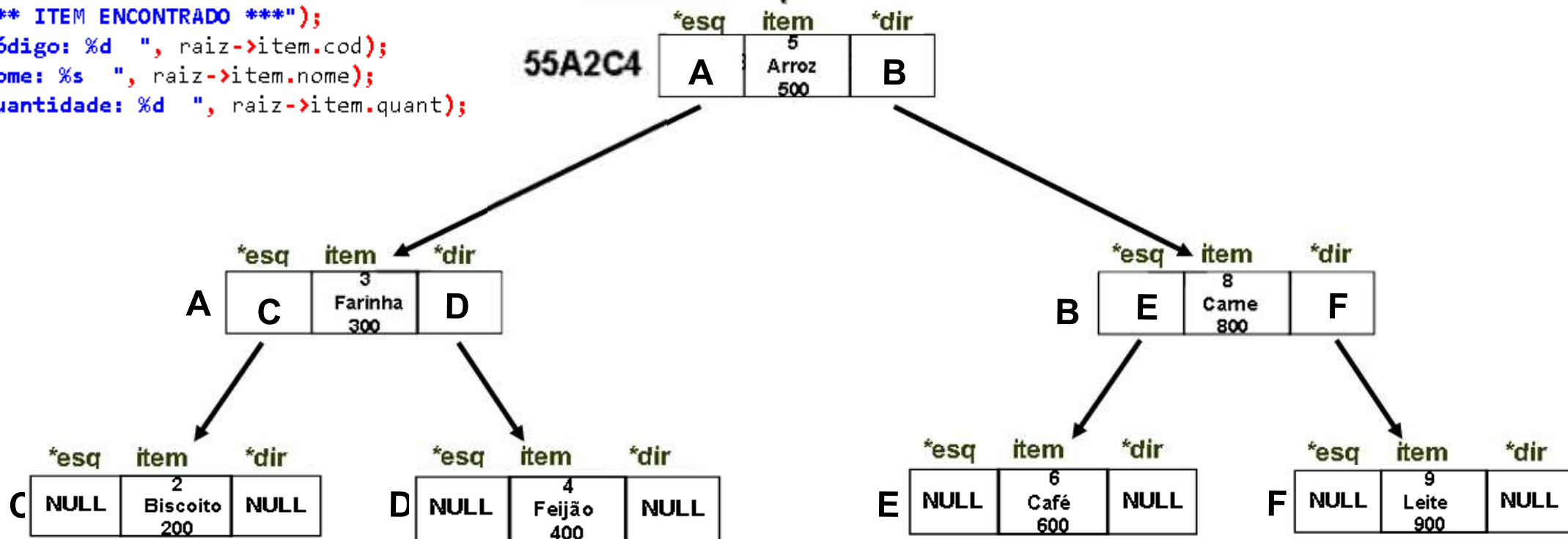
4

*rai

55A2C4

*raiz

55A2C4



```
void pesquisar(int codPesq, tipo_no *raiz){
```

```
    if(!raiz) {  
        printf("\n\n 0 item de código %d não está na árvore", codPesq);  
        return;  
    }
```

```
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }
```

```
    if(codPesq > raiz->item.cod)  
        pesquisar(codPesq, raiz->dir);
```

```
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }
```

cod

4

codpesq

4

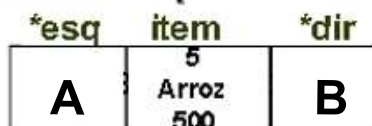
*rai

55A2C4

*raiz

55A2C4

55A2C4

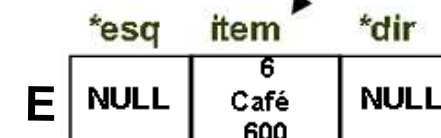
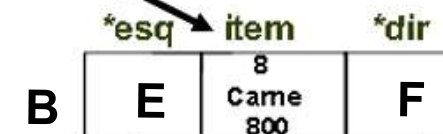
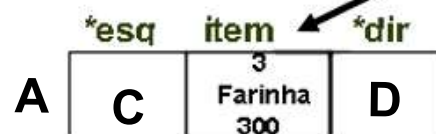


codpesq

4

*rai

Z A



```
void pesquisar(int codPesq, tipo_no *raiz){
```

```
    if(!raiz) {  
        printf("\n\n 0 item de código %d não está na árvore", codPesq);  
        return;  
    }
```

```
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }
```

```
    if(codPesq > raiz->item.cod)  
        pesquisar(codPesq, raiz->dir);
```

```
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }
```

cod

4

codpesq

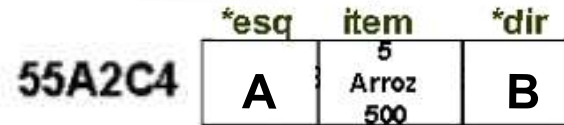
4

*rai

55A2C4

*raiz

55A2C4



codpesq

4

*rai

Z A

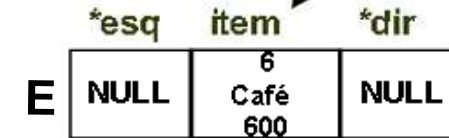
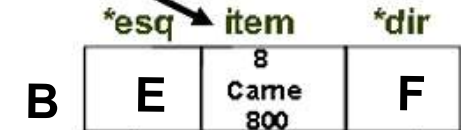


*rai

Z D

codpesq

4




```
void pesquisar(int codPesq, tipo_no *raiz){
```

```
    if(!raiz) {  
        printf("\n\n 0 item de código %d não está na árvore", codPesq);  
        return;  
    }
```

```
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }
```

```
    if(codPesq > raiz->item.cod)  
        pesquisar(codPesq, raiz->dir);
```

```
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }
```

cod

4

codpesq

4

*rai

55A2C4

*raiz

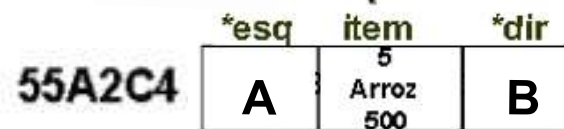
55A2C4

*** ITEM ENCONTRADO ***

Código: 4

Nome: Feijao

Quantidade: 400



codpesq

4

*rai

Z A

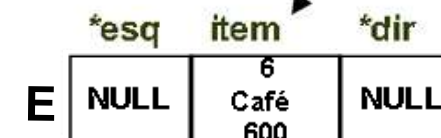


*rai

Z D

codpesq

4



```
void pesquisar(int codPesq, tipo_no *raiz){
```

```
    if(!raiz) {  
        printf("\n\n 0 item de código %d não está na árvore", codPesq);  
        return;  
    }
```

```
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }
```

```
    if(codPesq > raiz->item.cod){  
        pesquisar(codPesq, raiz->dir);  
    }
```

```
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }
```

cod

4

codpesq

4

*rai

55A2C4

*raiz

55A2C4

*** ITEM ENCONTRADO ***

Código: 4

Nome: Feijao

Quantidade: 400

*rai

Z A

codpesq

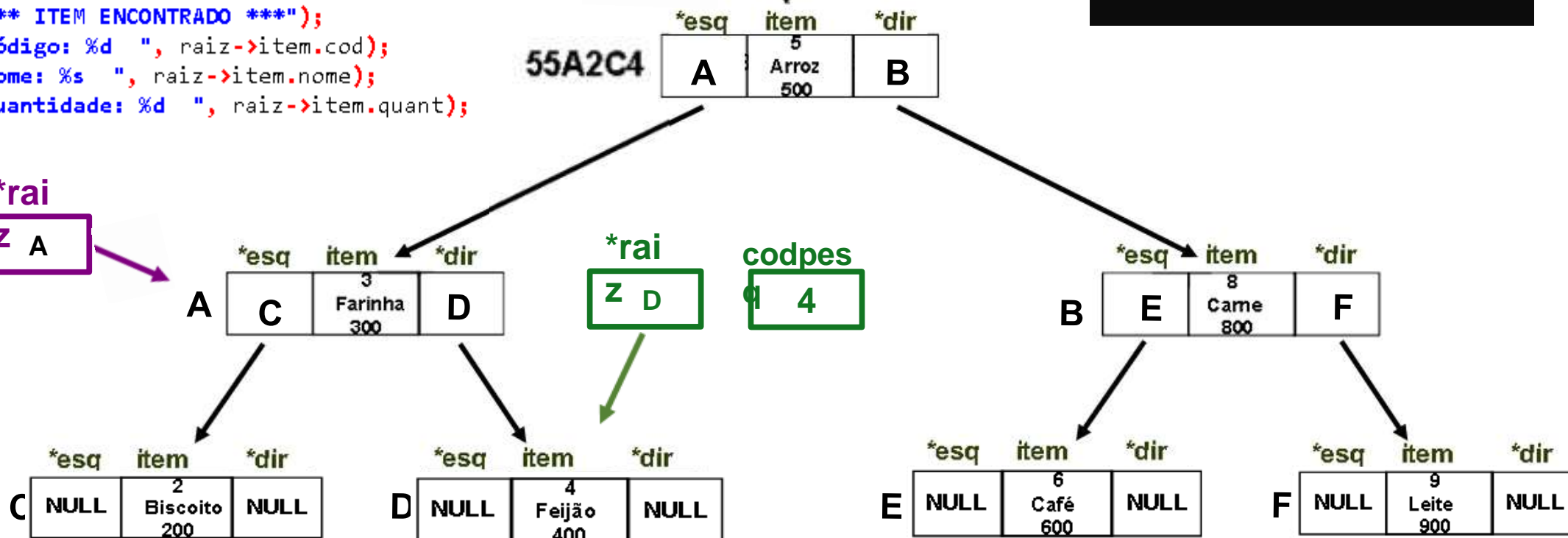
4

*rai

Z D

codpes

4



```
void pesquisar(int codPesq, tipo_no *raiz){
```

```
    if(!raiz) {  
        printf("\n\n 0 item de código %d não está na árvore", codPesq);  
        return;  
    }
```

```
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }
```

```
    if(codPesq > raiz->item.cod)  
        pesquisar(codPesq, raiz->dir);
```

```
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }
```

cod

4

codpesq

4

*rai

55A2C4

*raiz

55A2C4

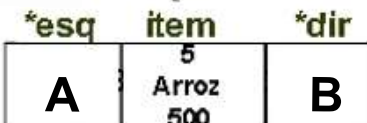
*** ITEM ENCONTRADO ***

Código: 4

Nome: Feijao

Quantidade: 400

55A2C4

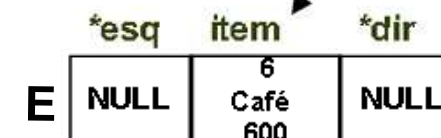
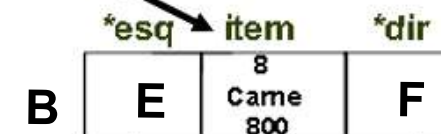
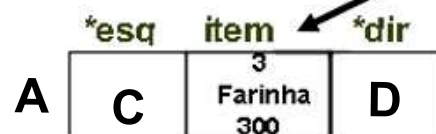


codpesq

4

*rai

Z A



```
void pesquisar(int codPesq, tipo_no *raiz){
```

```
    if(!raiz) {  
        printf("\n\n 0 item de código %d não está na árvore", codPesq);  
        return;  
    }
```

```
    if(codPesq < raiz->item.cod){  
        pesquisar(codPesq, raiz->esq);  
        return;  
    }
```

```
    if(codPesq > raiz->item.cod)  
        pesquisar(codPesq, raiz->dir);
```

```
    else{  
        printf("\n\n *** ITEM ENCONTRADO ***");  
        printf("\n\n Código: %d ", raiz->item.cod);  
        printf("\n\n Nome: %s ", raiz->item.nome);  
        printf("\n\n Quantidade: %d ", raiz->item.quant);  
    }
```

```
}
```

cod

4

codpesq

4

*rai

55A2C4

*raiz

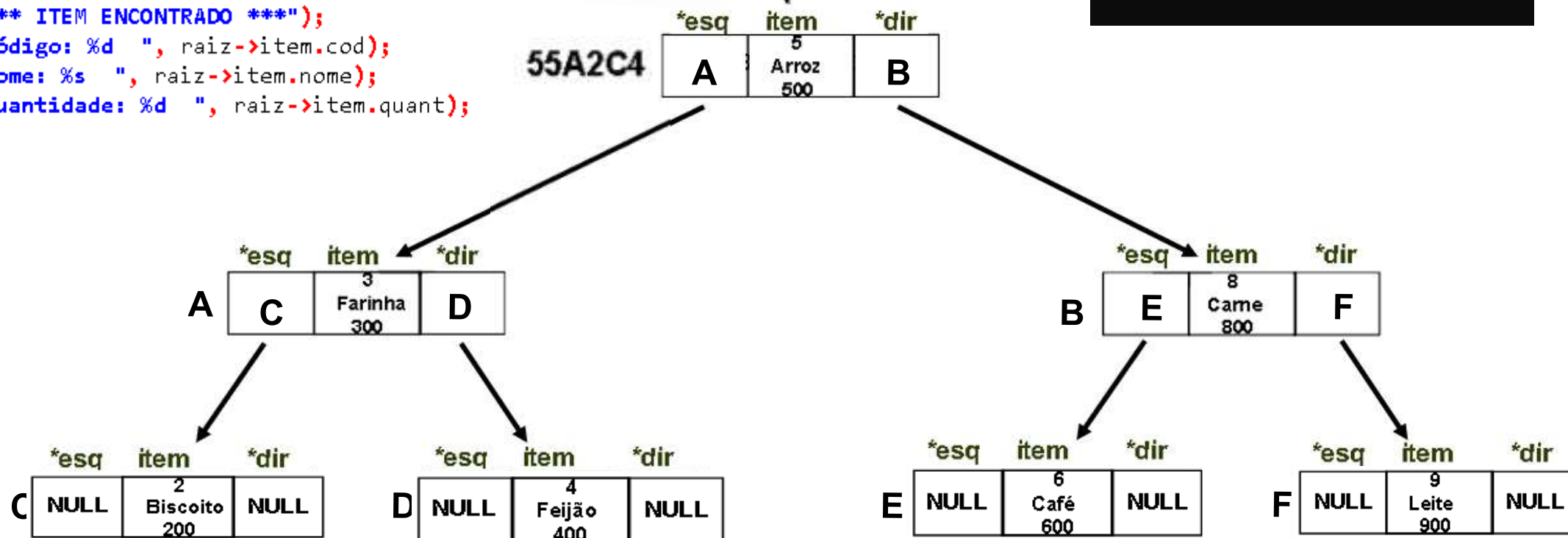
55A2C4

*** ITEM ENCONTRADO ***

Código: 4

Nome: Feijao

Quantidade: 400



cod

4

*** ITEM ENCONTRADO ***

Código: 4

Nome: Feijao

Quantidade: 400

