


Unidade VII: Árvore Binária - Inserção em C com ponteiro



PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

- Ponteiros
- Estrutura de arquivos
- makefile
- Arquivos “no”
- Arquivos “arvorebinaria”

- **Ponteiros** 
- Estrutura de arquivos
- makefile
- Arquivos “no”
- Arquivos “arvorebinaria”

Ponteiros

- Os ponteiros são apenas variáveis que armazenam o endereço de uma área de memória
- Da mesma forma que um int armazena inteiro e um double, número real , os ponteiros armazenam endereços de memória
- Os ponteiros possuem tipo, ou seja, temos ponteiro para endereços de memória de um int, de um float, de um char...

Declaração de Ponteiros

tipoPonteiro *nomeVariável;

- O asterisco na declaração de uma variável indica que essa não guardará um valor e sim um endereço para o tipo especificado

Operadores

- Operador endereço (**&**) determina o endereço de uma variável
- Operador de conteúdo de um ponteiro (*****) determina o conteúdo da posição de memória endereçada pelo ponteiro

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

Memória		Tela
	75h	
	...	
	CDh	

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;
```

```
int * y = & x;
```

```
printf("\n%i", x);
```

```
printf("\n%p", &x);
```

```
printf("\n%p", y);
```

```
printf("\n%p", &y);
```

```
printf("\n%i", *y);
```

x	Memória	
	10	75h
		...
		CDh

Tela

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;
```

```
int * y = &x;
```

```
printf("\n%i", x);
```

```
printf("\n%p", &x);
```

```
printf("\n%p", y);
```

```
printf("\n%p", &y);
```

```
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;
```

```
int * y = & x;
```

```
printf("\n%i", x);
```

```
printf("\n%p", &x);
```

```
printf("\n%p", y);
```

```
printf("\n%p", &y);
```

```
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```

int x = 10;

int * y = & x;

printf("\n%i", x);

printf("\n%p", &x);

printf("\n%p", y);

printf("\n%p", &y);

printf("\n%i", *y);

```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h
75h

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```

int x = 10;

int * y = & x;

printf("\n%i", x);

printf("\n%p", &x);

printf("\n%p", y);

printf("\n%p", &y);

printf("\n%i", *y);

```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h
75h
CDh

Exercício Resolvido (1)

- Faça o quadro de memória e mostre a saída na tela:

```

int x = 10;

int * y = & x;

printf("\n%i", x);

printf("\n%p", &x);

printf("\n%p", y);

printf("\n%p", &y);

printf("\n%i", *y);

```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h
75h
CDh
10

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 5;
int *y = &x;
int **z = &y;
printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);
```

	Memória	
x		75h
		...
y		A0h
		...
z		CDh

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 5;
int *y = &x;
int **z = &y;
printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);
```

Tela

	Memória	
x	5	75h
		...
y		A0h
		...
z		CDh

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 5;
int *y = &x;
int **z = &y;
printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);
```

Tela

Memória		
x	5	75h
		...
y	75h	A0h
		...
z		CDh

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 5;
int *y = &x;
int **z = &y;
printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);
```

Tela

	Memória	
x	5	75h
		...
y	75h	A0h
		...
z	A0h	CDh

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```

int x = 5;
int *y = &x;
int **z = &y;

printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);

```

Tela
x(5), &x(75h)

	Memória	
x	5	75h
		...
y	75h	A0h
		...
z	A0h	CDh

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```

int x = 5;
int *y = &x;
int **z = &y;
printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);

```

Tela
x(5), &x(75h)
y(75h), *y(5), &y(A0h)

	Memória	
x	5	75h
		...
y	75h	A0h
		...
z	A0h	CDh

Exercício Resolvido (2)

- Faça o quadro de memória e mostre a saída na tela:

```

int x = 5;
int *y = &x;
int **z = &y;
printf("\nx(%i), &x(%p)", x, &x);
printf("\ny(%p), *y(%i), &y(%p)", y, *y, &y);
printf("\nz(%p), &z(%p), *z(%p), **z(%i)", z, &z, *z, **z);

```

Tela

```

x(5), &x(75h)
y(75h), *y(5), &y(A0h)
z(A0h), &z(CDh), *z(75h), **z(5)

```

Memória

x	5	75h
		...
y	75h	A0h
		...
z	A0h	CDh

Alocar Memória em C: malloc

- Protótipo da função malloc()

void* malloc (int tamanho)

- O malloc aloca o número de bytes passados como parâmetro e retorna um ponteiro para a primeira posição da área alocada

Desalocar Memória em C: free()

- Protótipo da função free()

void free (void*)

- O free desaloca o espaço de memória apontado pelo ponteiro recebido como parâmetro

Exemplo do malloc() e do free()

```
char* p1 = (char*) malloc (sizeof(char));  
int* p2 = (int*) malloc (sizeof(int));  
float* p3 = (float*) malloc (sizeof(float));  
Cliente* p4 = (Cliente*) malloc (sizeof(Cliente));  
int* p5 = (int*) malloc (MAXTAM * sizeof (int));  
Cliente* p6 =(Cliente*) malloc (MAXTAM * sizeof (Cliente));  
free(p1);  
free(p2);  
free(p3);  
free(p4);  
free(p5);  
free(p6);
```


Alocar/Desalocar Memória em C++: new e delete

```
char* p1 = new char;
```

```
int* p2 = new int;
```

```
float* p3 = new float;
```

```
Cliente* p4 = new Cliente;
```

```
int* p5 = new int [MAXTAM];
```

```
Cliente* p6 = new Cliente[MAXTAM];
```

```
delete p1;
```

```
delete p2;
```

```
delete p3;
```

```
delete p4;
```

```
delete [ ] p5;
```

```
delete [ ] p6;
```

Exercícios Gráficos

em Java

Exercício Resolvido (3)

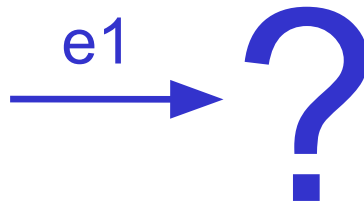
- Represente graficamente o código Java abaixo

```
Elemento e1;
```

Exercício Resolvido (3)

- Represente graficamente o código Java abaixo

Elemento e1;



Exercício Resolvido (3)

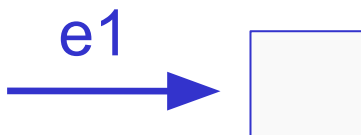
- Represente graficamente o código Java abaixo

```
Elemento e1 = new Elemento();
```

Exercício Resolvido (3)

- Represente graficamente o código Java abaixo

```
Elemento e1 = new Elemento();
```



Exercício Resolvido (3)

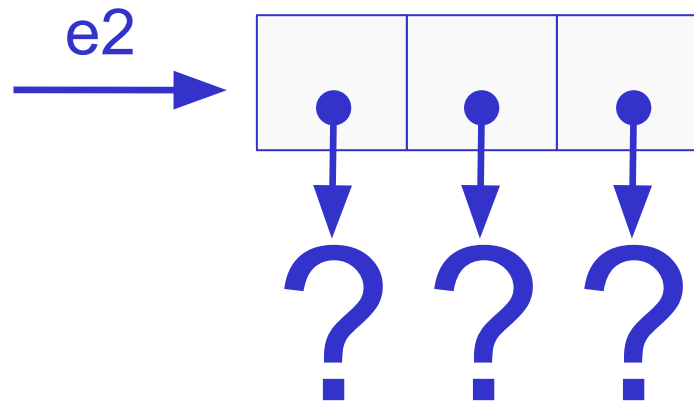
- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

Exercício Resolvido (3)

- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```



Exercício Resolvido (3)

- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

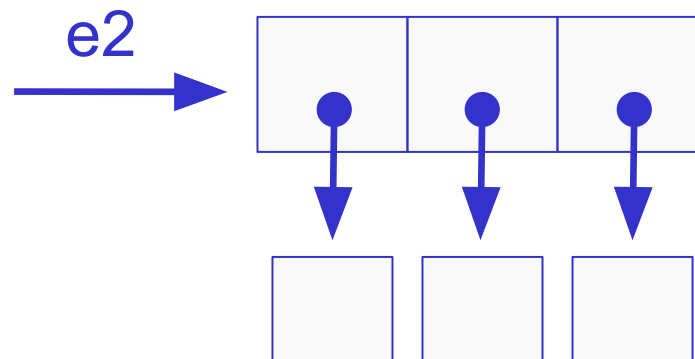
```
for (int i = 0; i < 3; i ++){  
    e2[i] = new Elemento();  
}
```

Exercício Resolvido (3)

- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

```
for (int i = 0; i < 3; i ++){  
    e2[i] = new Elemento();  
}
```



Exercícios Gráficos

em C

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento e1;
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

Elemento e1;

e1



Exercício Resolvido (4)

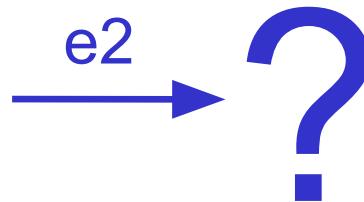
- Represente graficamente o código C abaixo

```
Elemento* e2;
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento* e2;
```



Exercício Resolvido (4)

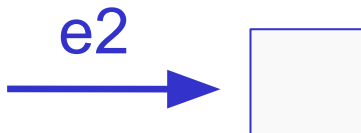
- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));
```


Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));
```



Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(3*sizeof(Elemento));
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(3*sizeof(Elemento));
```



Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento e3[3];
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

Elemento e3[3];



Exercício Resolvido (4)

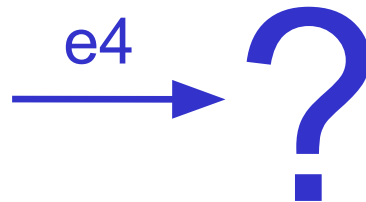
- Represente graficamente o código C abaixo

```
Elemento** e4;
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento** e4;
```



Exercício Resolvido (4)

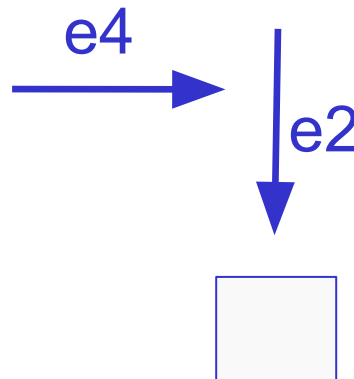
- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));  
Elemento** e4 = &e2;
```


Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));  
Elemento** e4 = &e2;
```



Exercício Resolvido (4)

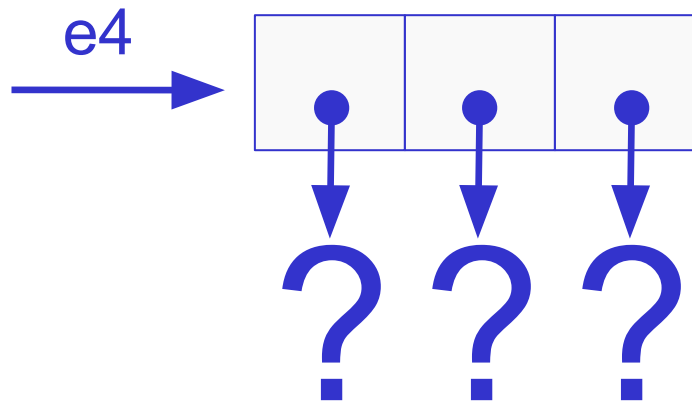
- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));
```



Exercício Resolvido (4)

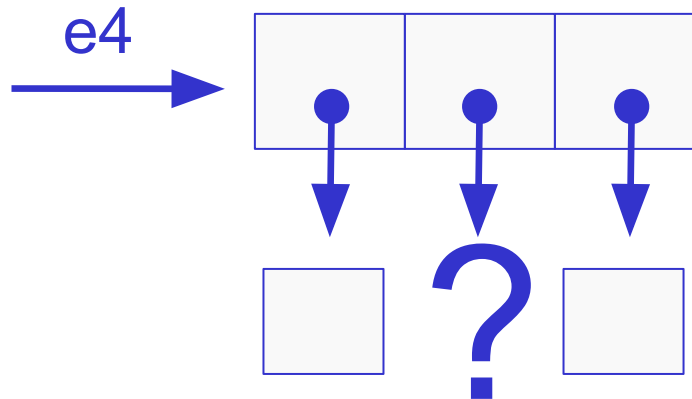
- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));  
e4[0] = (Elemento*) malloc(sizeof(Elemento*));  
e4[2] = (Elemento*) malloc(sizeof(Elemento*));
```

Exercício Resolvido (4)

- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));  
e4[0] = (Elemento*) malloc(sizeof(Elemento*));  
e4[2] = (Elemento*) malloc(sizeof(Elemento*));
```



Exercícios Gráficos

C++

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento e1;
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento e1;
```

e1



Exercício Resolvido (5)

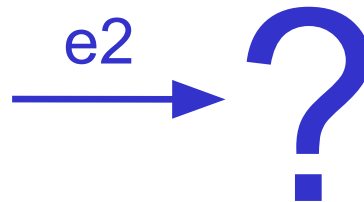
- Represente graficamente o código C++ abaixo

```
Elemento* e2;
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento* e2;
```



Exercício Resolvido (5)

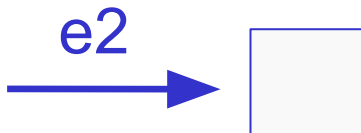
- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;
```



Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento[3];
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento[3];
```



Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento e3[3];
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento e3[3];
```



Exercício Resolvido (5)

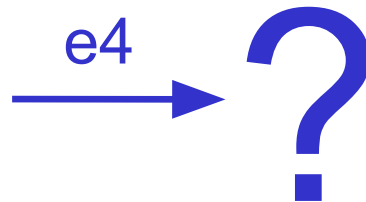
- Represente graficamente o código C++ abaixo

```
Elemento** e4;
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento** e4;
```



Exercício Resolvido (5)

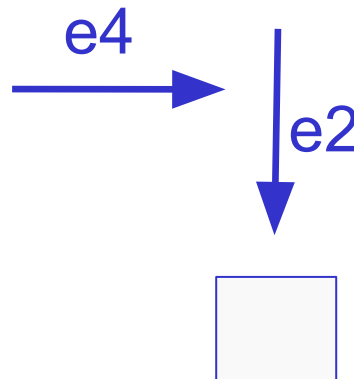
- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;  
Elemento** e4 = &e2;
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;  
Elemento** e4 = &e2;
```



Exercício Resolvido (5)

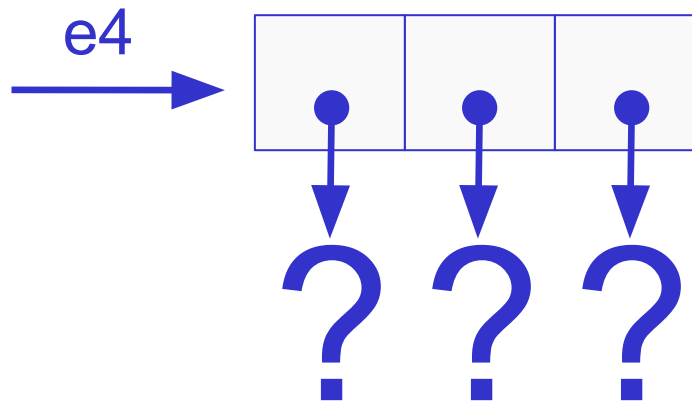
- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];
```



Exercício Resolvido (5)

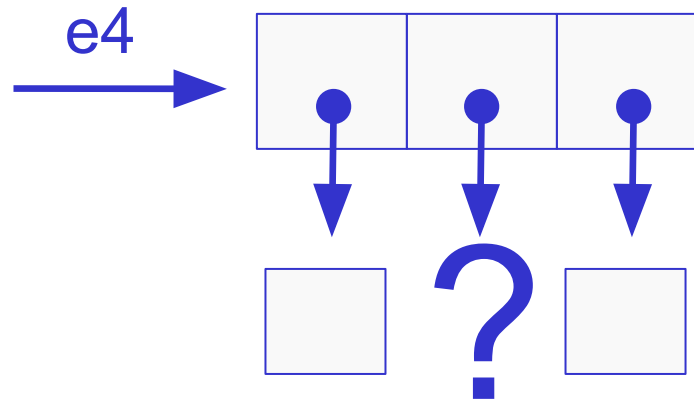
- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];  
e4[0] = new Elemento;  
e4[2] = new Elemento;
```

Exercício Resolvido (5)

- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];  
e4[0] = new Elemento;  
e4[2] = new Elemento;
```



Exercício Resolvido (6)

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}
```

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela**Memória**

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

```

```

int main(int argc, char *argv[]) {

```

```

    int a = 0, b = 0;

```

```

    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

Memória

a	0	33h
	...	
b	0	51h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

```

```

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

Memória

a	0	33h
	...	
b	0	51h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

Memória

a	0	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	0	C2h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela
Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	0	C2h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	1	C2h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}
```

```
int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}
```

Tela

(33h) (1) (1)

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	1	C2h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

```

```

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

(33h) (1) (1)

Memória

a	1	33h
	...	
b	0	51h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

```

```

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

```

(33h) (1) (1)
(1)(0)

```

Memória

a	1	33h
	...	
b	0	51h

- Faça o quadro de memória e mostre a saída na tela do programa acima

Exercício Resolvido (6)

```

void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

```

```

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}

```

Tela

```

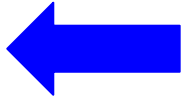
(33h) (1) (1)
(1)(0)

```

Memória


a	1	33h
	...	
b	0	51h

- Faça o quadro de memória e mostre a saída na tela do programa acima

- Ponteiros
- **Estrutura de arquivos** 
- makefile
- Arquivos “no”
- Arquivos “arvorebinaria”

Estrutura de Arquivos

- no.h
- no.c
- arvorebinaria.h
- arvorebinaria.c
- principal.c
- makefile

- Ponteiros
- Estrutura de arquivos
- **makefile** 
- Arquivos “no”
- Arquivos “arvorebinaria”

- Arquivo contendo um conjunto de diretivas usadas pela ferramenta de automação de compilação *make* para gerar um alvo / meta
- Nesse caso, os arquivos serão compilados digitando ***make***

```
1  all: exec
2
3  exec: principal.o arvorebinaria.o no.o
4      gcc -o exec principal.o arvorebinaria.o no.o
5
6  principal.o: principal.c
7      gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
8
9  arvorebinaria.o: arvorebinaria.c
10     gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
11
12 no.o: no.c
13     gcc -o no.o no.c -c -W -Wall -ansi -pedantic
14
15 clean:
16     rm -rf *.o *~ exec
17
18 limpa:
19     rm -rf *.o
```




```
1  all:
2
3  exec:
4      gcc -o exec principal.o arvorebinaria.o no.o
5
6  principal.o:
7      gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
8
9  arvorebinaria.o:
10     gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
11
12 no.o:
13     gcc -o no.o no.c -c -W -Wall -ansi -pedantic
14
15 clean:
16     rm -rf *.o *~ exec
17
18 limpa:
19     rm -rf *.o
```

comandos

```
graph LR
    C[comandos] --> L4[gcc -o exec principal.o arvorebinaria.o no.o]
    C --> L7[gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic]
    C --> L10[gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic]
    C --> L13[gcc -o no.o no.c -c -W -Wall -ansi -pedantic]
    C --> L16[rm -rf *.o *~ exec]
    C --> L19[rm -rf *.o]
```

```
1  all: exec
2
3  exec: principal.o arvorebinaria.o no.o
4
5
6  principal.o: principal.c
7
8
9  arvorebinaria.o: arvorebinaria.c
10
11
12  no.o: no.c
13
14
15  clean:
16
17
18  limpa:
19
```

pré-requisitos

```
graph LR
    PR[pré-requisitos] --> exec
    PR --> principal_o[principal.o]
    PR --> arvorebinaria_o[arvorebinaria.o]
    PR --> no_o[no.o]
```

```
1  all: exec
2
3  exec: principal.o arvorebinaria.o no.o
4      gcc -o exec principal.o arvorebinaria.o no.o
5
6  principal.o: principal.c
7      gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
8
9  arvorebinaria.o: arvorebinaria.c
10     gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
11
12 no.o: no.c
13     gcc -o no.o no.c -c -W -Wall -ansi -pedantic
14
15 clean:
16     rm -rf *.o *~ exec
17
18 limpa:
19     rm -rf *.o
```

Exercício Resolvido (7)

- Na pasta da árvore em C, digite a sequência de comandos abaixo e explique a saída
 - 1) make all ; ls
 - 2) make clean ; ls
 - 3) make ; ls
 - 4) make clean ; ls
 - 5) make exec ; ls
 - 6) make limpa ; ls
 - 7) make no.o ; ls

Exercício Resolvido (7)

- Na pasta da árvore em C, digite a sequência de comandos abaixo e explique a saída

- 1) **make all ; ls**
- 2) **make clean ; ls**
- 3) **make ; ls**
- 4) **make clean ; ls**
- 5) **make exec ; ls**
- 6) **make limpa ; ls**
- 7) **make no.o ; ls**

```
:$ make all ; ls
```

```
gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic  
gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic  
gcc -o no.o no.c -c -W -Wall -ansi -pedantic  
gcc -o exec principal.o arvorebinaria.o no.o
```

```
arvorebinaria.c arvorebinaria.h arvorebinaria.o exec makefile  
no.c no.h no.o principal.c principal.o
```

```
:$ make clean ; ls
```

```
rm -rf *.o *~ exec
```

```
arvorebinaria.c arvorebinaria.h makefile no.c no.h principal.c
```

Exercício Resolvido (7)

- Na pasta da árvore em C, digite a sequência de comandos abaixo e explique a saída

- 1) `make all ; ls`
- 2) `make clean ; ls`
- 3) `make ; ls`
- 4) `make clean ; ls`
- 5) `make exec ; ls`
- 6) `make limpa ; ls`
- 7) `make no.o ; ls`

```
:$ make ; ls
```

```
gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic  
gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic  
gcc -o no.o no.c -c -W -Wall -ansi -pedantic  
gcc -o exec principal.o arvorebinaria.o no.o
```

```
arvorebinaria.c arvorebinaria.h arvorebinaria.o exec makefile  
no.c no.h no.o principal.c principal.o
```

```
:$ make clean ; ls
```

```
rm -rf *.o *~ exec
```

```
arvorebinaria.c arvorebinaria.h makefile no.c no.h principal.c
```

Exercício Resolvido (7)

- Na pasta da árvore em C, digite a sequência de comandos abaixo e explique a saída

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) make clean ; ls
- 5) **make exec ; ls**
- 6) **make limpa ; ls**
- 7) make no.o ; ls

```
:$ make exec ; ls
```

```
gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic  
gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic  
gcc -o no.o no.c -c -W -Wall -ansi -pedantic  
gcc -o exec principal.o arvorebinaria.o no.o
```

```
arvorebinaria.c arvorebinaria.h arvorebinaria.o exec makefile  
no.c no.h no.o principal.c principal.o
```

```
:$ make limpa ; ls
```

```
rm -rf *.o *
```

```
arvorebinaria.c arvorebinaria.h exec makefile no.c no.h  
principal.c
```


Exercício Resolvido (7)

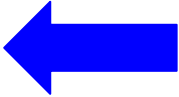
- Na pasta da árvore em C, digite a sequência de comandos abaixo e explique a saída

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) make clean ; ls
- 5) make exec ; ls
- 6) make limpa ; ls
- 7) **make no.o ; ls**

```
:$ make no.o ; ls
```

```
gcc -o no.o no.c -c -W -Wall -ansi -pedantic
```

```
arvorebinaria.c arvorebinaria.h arvorebinaria.o exec makefile  
no.c no.h no.o principal.c principal.o
```

- Ponteiro
- Estrutura de arquivos
- makefile
- **Arquivos “no”** 
- Arquivos “arvorebinaria”


Arquivos “no”

//no.h

```
typedef struct No {  
    int elemento;  
    struct No *esq, *dir;  
} No;  
  
No* novoNo(int);
```

//no.c

```
#include <stdlib.h>  
#include "no.h"  
  
No* novoNo(int elemento) {  
    No* novo = (No*) malloc(sizeof(No));  
    novo->elemento = elemento;  
    novo->esq = NULL;  
    novo->dir = NULL;  
    return novo;  
}
```

- Ponteiro
- Estrutura de arquivos
- makefile
- Arquivos “no”
- **Arquivos “arvorebinaria”** 

Arquivos “arvorebinaria”

```
//arvorebinaria.h
#include "no.h"
#define bool short
#define true 1
#define false 0

bool pesquisarRec(int, No*);
void caminharCentralRec(No*);
void caminharPreRec(No*);
void caminharPosRec(No*);
void inserirRec(int, No**);
void removerRec(int, No**);
void maiorEsq(No**, No**);
```

```
void start();
```

```
bool pesquisar(int);
void caminharCentral();
void caminharPre();
void caminharPos();
void inserir(int);
void remover(int);
```

```
//arvorebinaria.c
#include "no.h"
#include <err.h>
#include <stdlib.h>
#include <stdio.h>
#include "arvorebinaria.h"
```

```
No* raiz;
```

```
void start() {
    raiz = NULL;
}
```



Arquivos “arvorebinaria”

```
//arvorebinaria.h
#include "no.h"
#define bool short
#define true 1
#define false 0

bool pesquisarRec(int, No*);
void caminharCentralRec(No*);
void caminharPreRec(No*);
void caminharPosRec(No*);
void inserirRec(int, No**);
void removerRec(int, No**);
void maiorEsq(No**, No**);

void start();
bool pesquisar(int);
void caminharCentral();
void caminharPre();
void caminharPos();
void inserir(int);
void remover(int);
```

Como o C tem apenas a passagem de parâmetros por valor, neste material, optamos por fazer a inserção usando o endereço de ponteiro

Poderíamos, também, usar as duas estratégias implementadas em nosso código Java

Implementação da Função Inserir

- Anteriormente, em Java, apresentamos duas implementações do inserir()

```
No inserir(int x, No i) //Java
```

```
void inserir(int x, No i, No pai) //Java
```

- As implementações correspondentes em C seriam, respectivamente:

```
No* inserir(int x, No* i) //C
```

```
void inserir(int x, No* i, No* pai) //C
```

Implementação da Função Inserir

- Anteriormente, em Java, apresentamos duas implementações do inserir()

No inserir(int x, No i) //Java

void inserir(int x, No i, No pai) //Java

- As implementações correspondentes em C seriam, respectivamente:

No* inserir(int x, No* i) //C

void inserir(int x, No* i, No* pai) //C

Primeira Opção para o Inserir em C/Java

//código em Java

```
void inserir(int x) {  
    raiz = inserir(x, raiz);  
}
```

```
No inserir(int x, No i) {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new ("Erro!");  
    }  
    return i;  
}
```

//código em C

```
void inserir(int x) {  
    raiz = inserirRec(x, raiz);  
}  
  
No* inserirRec(int x, No* i) {  
    if (i == NULL) {  
        i = novoNo(x);  
    } else if (x < i->elemento) {  
        i->esq = inserirRec(x, i->esq);  
    } else if (x > i->elemento) {  
        i->dir = inserirRec(x, i->dir);  
    } else {  
        errx(1, "Erro ao inserir!");  
    }  
    return i;  
}
```

Implementação da Função Inserir

- Anteriormente, em Java, apresentamos duas implementações do inserir()

No inserir(int x, No i) //Java

void inserir(int x, No i, No pai) //Java

- As implementações correspondentes em C seriam, respectivamente:

No* inserir(int x, No* i) //C

void inserir(int x, No* i, No* pai) //C

Segunda Opção para o Inserir em C/Java

//código em Java

```
void inserirPai(int x) {
    if (raiz == null) {
        raiz = new No(x);
    } else if (x < raiz.elemento) {
        inserirPai(x, raiz.esq, raiz);
    } else if (x > raiz.elemento) {
        inserirPai(x, raiz.dir, raiz);
    } else { throw new("Erro!");
    } }
}
```

```
void inserirPai(int x, No i, No pai) {
    if (i == null) {
        if (x < i.elemento){ pai.esq = new No(x);
        } else { pai.dir = new No(x); }
    } else if (x < i.elemento) {
        inserirPai(x, i.esq, i);
    } else if (x > i.elemento) {
        inserirPai(x, i.dir, i);
    } else { throw new("Erro!");
    } }
}
```

//código em C

```
void inserirPai(int x) {
    if(raiz == NULL){
        raiz = novoNo(x);
    } else if(x < raiz->elemento){
        inserirPaiRec(x, raiz->esq, raiz);
    } else if(x > raiz->elemento){
        inserirPaiRec(x, raiz->dir, raiz);
    } else { errx(1, "Erro ao inserir!");
    } }
}
```

```
void inserirPaiRec(int x, No* i, No* pai) {
    if (i == NULL) {
        if(x < i->elemento){ pai->esq = novoNo(x);
        } else { pai->dir = novoNo(x); }
    } else if (x < i->elemento) {
        inserirPaiRec(x, i->esq, i);
    } else if (x > i->elemento) {
        inserirPaiRec(x, i->dir, i);
    } else { errx(1, "Erro ao inserir!");
    } }
}
```

Arquivos “arvorebinaria”

```
//arvorebinaria.h
```

```
#include "no.h"
```

```
#define bool short
```

```
#define true 1
```

```
#define false 0
```

```
bool pesquisarRec(int, No*);
```

```
void caminharCentralRec(No*);
```

```
void caminharPreRec(No*);
```

```
void caminharPosRec(No*);
```

```
void inserirRec(int, No**);
```

```
void removerRec(int, No**);
```

```
void maiorEsq(No**, No**);
```

```
void start();
```

```
bool pesquisar(int);
```

```
void caminharCentral();
```

```
void caminharPre();
```

```
void caminharPos();
```

```
void inserir(int);
```

```
void remover(int);
```

```
//arvorebinaria.c
```

■ ■ ■

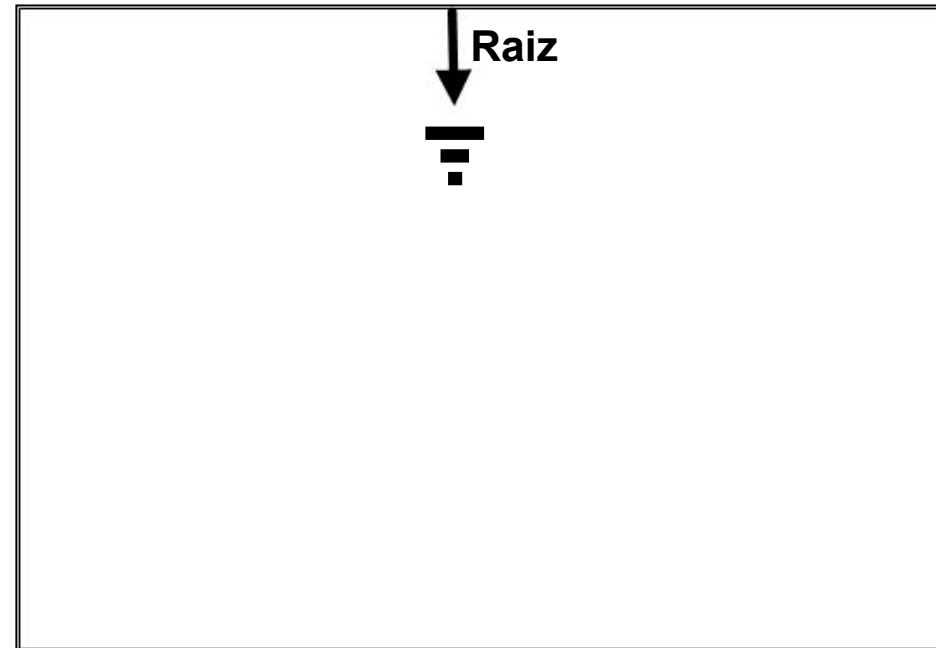
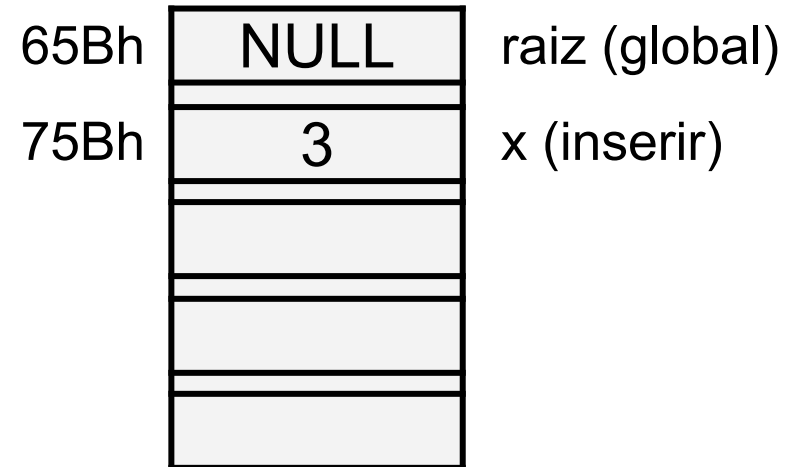
```
void inserir(int x) {
    inserirRec(x, &raiz);
}
```

```
void inserirRec(int x, No** i) {
    if (*i == NULL) {
        *i = novoNo(x);
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

Arquivos “arvorebinaria”

```
//arvorebinaria.c (supondo inserir o 3)
```

```
void inserir(int x) {  
    inserirRec(x, &raiz);  
}  
  
void inserirRec(int x, No** i) {  
    if (*i == NULL) {  
        *i = novoNo(x);  
    } else if (x < (*i)->elemento) {  
        inserirRec(x, &((*i)->esq));  
    } else if (x > (*i)->elemento) {  
        inserirRec(x, &((*i)->dir));  
    } else {  
        errx(1, "Erro ao inserir!");  
    }  
}
```



Arquivos “arvorebinaria”

//arvorebinaria.c (supondo inserir o 3)

```
void inserir(int x) {
    inserirRec(x, &raiz);
}
```

```
void inserirRec(int x, No** i) {
    if (*i == NULL) {
        *i = novoNo(x);
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

65Bh

75Bh



raiz (global)

x (inserir)

Raiz



Arquivos “arvorebinaria”

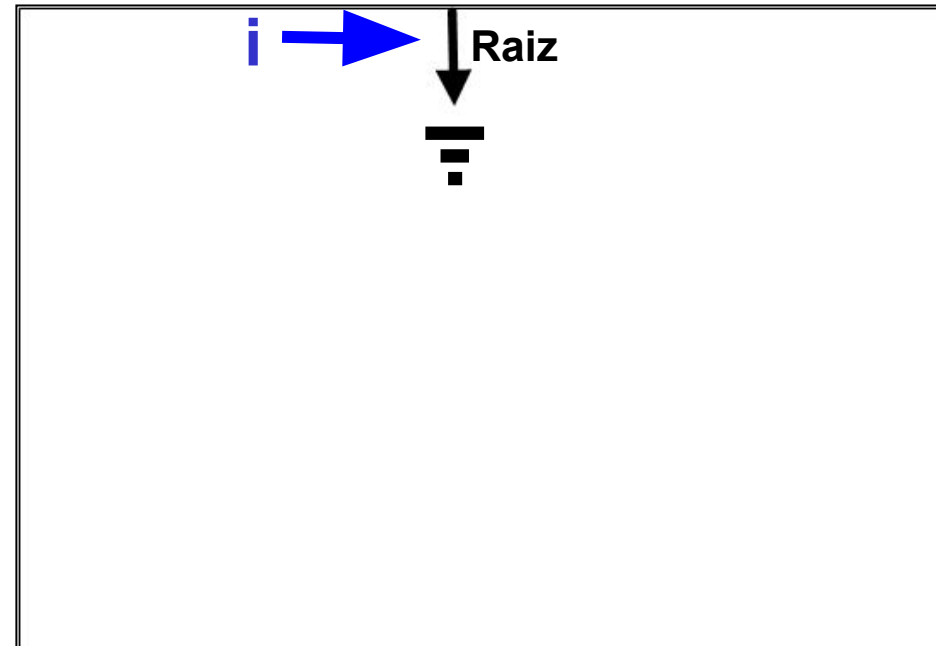
//arvorebinaria.c (supondo inserir o 3)

```
void inserir(int x) {
    inserirRec(x, &raiz);
}
```

```
void inserirRec(int x, No** i) {
```

```
    if (*i == NULL) {
        *i = novoNo(x);
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

65Bh	NULL	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)



Arquivos “arvorebinaria”

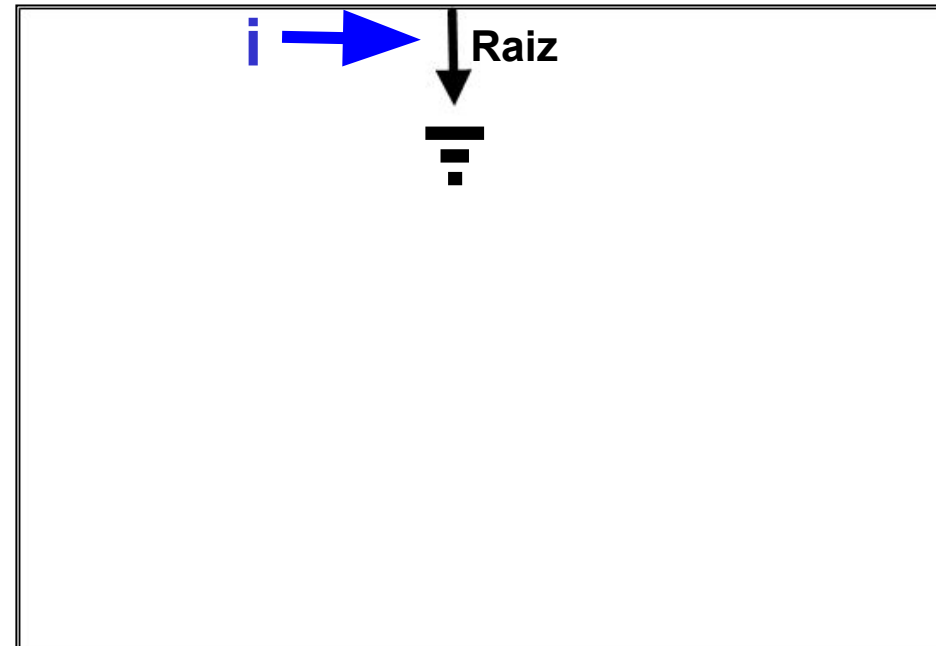
//arvorebinaria.c (supondo inserir o 3)

```
void inserir(int x) {
    inserirRec(x, &raiz);
}
```

```
void inserirRec(int x, No** i) {
    if (*i == NULL) {
```

```
        *i = novoNo(x);
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

65Bh	NULL	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)



Arquivos “arvorebinaria”

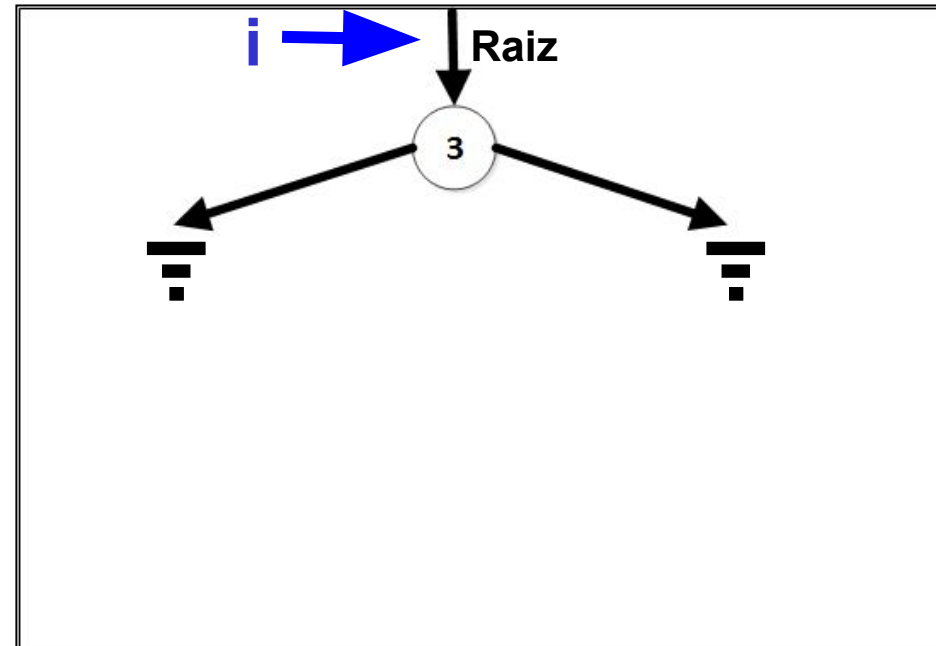
//arvorebinaria.c (supondo inserir o 3)

```
void inserir(int x) {
    inserirRec(x, &raiz);
}
```

```
void inserirRec(int x, No** i) {
    if (*i == NULL) {
        *i = novoNo(x);
```

```
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

65Bh	922h	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)
922h	3/null/null	(novoNo)



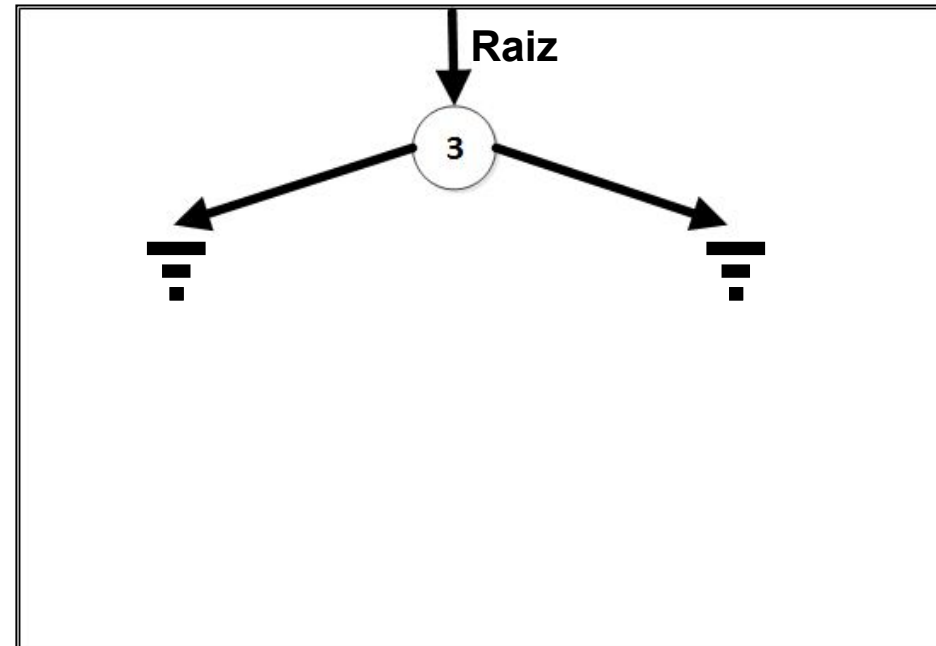
Arquivos “arvorebinaria”

//arvorebinaria.c (supondo inserir o 3)

```
void inserir(int x) {
    inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
    if (*i == NULL) {
        *i = novoNo(x);
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

65Bh	922h	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)
922h	3/null/null	(novoNo)



Arquivos “arvorebinaria”

//arvorebinaria.c (supondo inserir o 3)

```
void inserir(int x) {
    inserirRec(x, &raiz);
}
```

```
void inserirRec(int x, No** i) {
    if (*i == NULL) {
        *i = novoNo(x);
    } else if (x < (*i)->elemento) {
        inserirRec(x, &((*i)->esq));
    } else if (x > (*i)->elemento) {
        inserirRec(x, &((*i)->dir));
    } else {
        errx(1, "Erro ao inserir!");
    }
}
```

65Bh	922h	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)
922h	3/null/null	(novoNo)

