



Árvore Binária em C

[Ponteiros](#)

[Operadores](#)

[Alocar Memória em C](#)

[Desalocar Memória em C](#)

[Registro vs Ponteiro](#)

[Exercícios Gráficos](#)

[Arquivos "no"](#)

[no.h](#)

[no.c](#)

[Arquivos "arvorebinaria"](#)

[arvorebinaria.h](#)

[arvorebinaria.c](#)

[Inserir em C - 1 Opção](#)

[Inserir em C - 2 Opção](#)

Ponteiros

- Variáveis que armazenam **endereço de memória**

Operadores

- & - Endereço
- * - Conteúdo de um Ponteiro

Alocar Memória em C

- 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

- void free(void*)
- O free desaloca espaço de memória apontado pelo ponteiro recebido como parâmetro

Registro vs Ponteiro

- Acessamos um atributo de um registro fazendo **registro.atributo**
- Acessamos um atributo de um registro apontado por um ponteiro fazendo **ponteiro->atributo**

```
Cliente registro;
Cliente *ponteiro = (Cliente*)malloc(sizeof(Cliente));
registro.codigo = 1;
strcpy(registro.nome, "AA");
ponteiro -> codigo = 2;
strcpy(ponteiro->nome, "BB");
```

Exercícios Gráficos

- Disponíveis nesse repositório

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;
}
```

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;
}
```

Inserir em C - 1 Opção

```
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;  
}
```

Inserir em C - 2 Opção

```
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<pai->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!");  
    }  
}
```