

Ponteiros Nulos e Vazios

Até o momento, temos estudado vários tipos de ponteiros que apontam para outros lugares e assim por diante. Mas, C e C++ possuem outras formas de ponteiros, e, não obrigatoriamente, um ponteiro tem que apontar para algum lugar, pois um ponteiro simplesmente pode apontar para lugar nenhum.

Null Pointers (Ponteiros Nulos)

Nem sempre um ponteiro deve apontar para algum lugar, principalmente se este lugar ainda não existir. Então, é possível um ponteiro ser nulo.

Para um ponteiro apontar para lugar algum devemos lhe atribuir o valor inteiro 0, ou a constante NULL. Não importa se o ponteiro for do tipo float, int, char, etc., pois automaticamente haverá um typecasting para o tipo de dado correto.

É importante entender este conceito de ponteiro nulo porque facilita o trabalho com estruturas de memória dinâmicas como listas encadeadas, filas, arvores binárias, etc. Pois, na maior parte do tempo existirá um ponteiro, mas não um endereço físico já criado na memória.

```
#include <iostream>
using namespace std;

int main (void) {
    int *nullPointer;
    nullPointer = NULL;
    cout << nullPointer << endl;
    system ("pause");
    return EXIT_SUCCESS;
}
```

Void Pointers (Ponteiros Vazios)

Chamamos de ponteiros vazios qualquer ponteiro que não possua um tipo de dado específico, ou seja, se seu tipo de dado for void.

Ponteiros vazios ou void pointers são ponteiros genéricos que podem apontar para qualquer outro tipo de dado, mas trazem consigo problemas lógicos.

Tais problemas residem no fato de que como são ponteiros vazios - não há um tipo de dado específico - então, não podemos determinar seu tamanho, assim como também não podemos dereferenciá-lo diretamente.

Esses problemas são facilmente resolvidos se induzirmos o endereço void para o endereço do tipo de dado correto, da mesma forma que executamos qualquer typecast.

Entender o conceito de void pointers (ponteiros vazios) é de suma importância para a compreensão de estruturas dinâmicas de memória, porque trabalharemos o tempo todo com funções que retornam endereços de memória genéricos (que podem ser qualquer tipo de endereço de memória, portanto sem tipo de dado - void).

```
#include <iostream>
using namespace std;

int main (void) {
    void *ptr;
    int *ptrVar1, var1 = 10;
    float *ptrVar2, var2 = 50.123;
    ptr = &var1; // ponteiro aponta para um inteiro
    ptrVar1 = (int*) ptr; // typecasting: ponteiro genérico p/ inteiro
    cout << *ptrVar1 << endl;
    ptr = &var2; // o mesmo ponteiro aponta agora para um float
    ptrVar2 = (float*) ptr; // typecasting: ponteiro genérico p/ float
    cout << *ptrVar2 << endl;
    system ("pause");
}
```



```
return EXIT_SUCCESS;  
}
```

Como demonstrado no exemplo acima, ponteiros vazios podem apontar para qualquer tipo de dado (inteiro, float, char, etc), porém não podem ser dereferenciados diretamente. Para contornar este problema, usamos typecast para que um ponteiro concreto possa ser dereferenciado no lugar do ponteiro genérico.



Autor: Denys William Xavier

Este artigo está sob Licença Creative Commons

Para ver uma cópia desta licença, visite <http://creativecommons.org/licenses/by-nc-sa/2.5/br/> ou envie uma carta para Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.