Estrutura de Programação em C.

Número ISBN: 978-85-917609-2-3

Agradecemos a Deus, aos mestres que fizeram parte de nossa formação ao nossos pais, especialmente ao pai e amigo Pedro Marconi que não está mais entre nós, aos nossos amigos as nossas companheiras e a todos que colaboraram diretamente e indiretamente para conclusão desta obra .

Coleção “ Programação C ”.

Guilherme Marconi

Ricardo Akira Harada

Prefixo Editorial: 917609



BIBLIOTECA
NACIONAL



ISBN do Anais do ENEAC 2013: 978-85-7426-129-4

1.1 Símbolos usados com ponteiros

Para usar ponteiros, são utilizados os operadores * e &. Estes símbolos possuem diferentes significados na linguagem C, dependendo do contexto. Portanto, um mesmo símbolo pode representar diferentes conceitos. O símbolo * em uma expressão aritmética representa multiplicação. Este símbolo, quando usado na declaração de variáveis, especifica uma variável do tipo ponteiro. Já em comandos de atribuição, este símbolo antecedendo uma variável refere-se a um ponteiro, o qual indica o conteúdo do endereço apontado. De forma similar, o símbolo & pode representar o operador bit-a-bit ou pode representar o endereço de uma variável.

1.2 Aritmética de ponteiros

Ponteiros armazenam endereços de memória. Com ponteiros, pode-se usar os operadores aritméticos de adição e de subtração. Estes operadores aritméticos modificam o conteúdo de um ponteiro, aumentando ou diminuindo o valor do endereço armazenado. O trecho de código a seguir exemplifica o uso de operadores aritméticos com ponteiros. O ponteiro pont_idade aponta para uma variável do tipo int. Supondo que o tipo int ocupe 2 bytes, cada vez que pont_idade é incrementado, seu endereço é acrescido do tamanho de 2 bytes. Este é um aspecto crucial na aritmética de ponteiros. O incremento e o decremento do conteúdo de ponteiros é feito de acordo com o tamanho do tipo base apontado pelo ponteiro. Desta forma, deve-se sempre indicar o tipo apontado por um ponteiro na sua declaração. Em ocasiões especiais, pode-se declarar um ponteiro do tipo void (i.e., declarado como void *pont_void). Este tipo de ponteiro pode ser usado para apontar para diferentes tipos de dados em um mesmo programa. Porém, o controle do aumento e do decremento de um ponteiro void deve ser realizado sem usar aritmética de ponteiros.

```
...
int *pont_idade; // ponteiro do tipo int
...
pont_idade = &idade; // assuma que endereço de idade é 704
pont_idade++; // conteúdo de pont_idade recebe 706
pont_idade+=3; // conteúdo de pont_idade recebe 712
...
```

1.3 Declaração de ponteiros com inicialização

Um ponteiro pode ser inicializado juntamente com a sua declaração. O trecho abaixo inicializa o ponteiro pont_idade com o endereço de memória 704. Note que, em uma declaração de variáveis, *pont_idade não se refere ao conteúdo do endereço apontado. Em contrapartida, em uma linha de comando, tal como em idade = *pont_idade, *pont_idade refere-se ao conteúdo do endereço apontado.

```
...
int *pont_idade = 704; // declaração e inicialização
...
idade = *pont_idade;
...
```

Atribuição de ponteiros

Um ponteiro pode receber o valor de outro ponteiro de forma similar ao que ocorre em um comando de atribuição com outros tipos de variáveis. Isto significa que um ponteiro receberá o endereço de memória armazenado em outro ponteiro. O trecho a seguir ilustra um exemplo de atribuição. Como resultado final, pont_idade2 irá armazenar o endereço 704. Portanto, após o comando de atribuição, ambos os ponteiros apontam para o mesmo endereço.

```
...
int *pont_idade1 = 704;
int *pont_idade2;
...
pont_idade2 = pont_idade1;
...
```

Comparação de ponteiros

De forma similar à atribuição de ponteiros, ponteiros podem ser comparados entre si usando operadores relacionais e de igualdade. Neste caso, compara-se os endereços armazenados nos ponteiros. O trecho a seguir ilustra a comparação de dois ponteiros usando o operador relacional > para decidir a execução de um trecho de programa.

```
...
if (pont_idade1 > pont_idade2)
{
// seqüência de comandos
}
...
```

Ponteiro nulo

Quando um ponteiro não aponta para nenhum endereço válido, o ponteiro é inicializado com o endereço 0. Esta é uma convenção usada por programadores C. Para facilitar o seu uso, a linguagem C define a constante NULL para representar o endereço 0. O trecho de programa a seguir ilustra a inicialização do ponteiro pont_idade com o valor NULL. Sempre que um ponteiro ficar temporariamente sem uso, deve-se atribuir NULL a ele. Caso contrário, erros podem acontecer durante a execução do programa, os quais podem inclusive travar o computador. Por exemplo, um ponteiro que armazena um endereço arbitrário pode estar apontando para uma área de memória que contém o próprio conjunto de instruções do programa. A modificação do conteúdo apontado por este ponteiro irá causar a perda da instrução e, por conseguinte, a execução incorreta do programa ou o travamento do programa. Previne-se este erro atribuindo o valor NULL ao ponteiro sem uso.

```
...
int *pont_idade;
...
pont_idade = NULL; // pont_idade = 0
```