PONTEIROS

Este tutorial tem como base os livros Fundamentos da Programação de Computadores: Algoritmos, Pascal, C/C++ e Treinamento em Linguagem C.

1. Endereços de memória;
2. Ponteiros;
3. Ponteiros – tamanho e endereçamento;
4. Ponteiros e vetores;
5. Passagem de parâmetros por referência.

O objetivo desta aula é conhecer o conceito de ponteiro e sua aplicação em algoritmos computacionais; entender como um dado é acessado na memória; e sua relação com vetores e funções.

TEMA 1 – ENDEREÇOS DE MEMÓRIA

A memória de um computador é dividida em bytes, numerados de zero até o limite de memória da máquina. Esses números são chamados endereços de bytes, usados como referências, pelo computador, para localizar as variáveis(Mizrahi, 2008).

Toda variável tem uma localização na memória, e o endereço de identificação desta variável é o primeiro byte ocupado por ela, conforme explicado anteriormente.

Se o programa contém a informação somente do endereço do primeiro byte da variável, como ele sabe quais endereços deve ler? Bom, para obter esta resposta, o programa deve saber que toda variável está armazenada em bytes sequenciais e, identificando o tamanho desta variável pelo seu tipo, infere até onde a leitura dos endereços deve ir.

Por exemplo, uma variável do tipo int em C com tamanho 4 bytes, ou seja, sabendo o endereço inicial, sabe que, a partir, dele temos mais três endereços sequenciais que correspondem à variável desejada.

Quando o programa é carregado na memória, ocupa certa quantidade de bytes, e toda variável e função desse programa terão seu espaço e endereço particular. Para conhecer o endereço em que uma variável está alocada, usa-se o operador de endereços &. Observe a Figura 1, que mostra um algoritmo que vai imprimir os endereços de três variáveis usando a função printf() nas linhas 9, 10 e 11.

Figura 1 – Algoritmo que imprime o endereço de três variáveis

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

    int x, y, z;

    /* %p para ponteiros */
    printf("O endereco de x: %p \n", &x);
    printf("O endereco de y: %p \n", &y);
    printf("O endereco de z: %p \n\n", &z);

    system("pause");
    return 0;
}
```

A Figura 2 mostra a saída do algoritmo acima após a sua execução.

Figura 2 – Saída do algoritmo

```
O endereco de x: 000000000062FE4C
O endereco de y: 000000000062FE48
O endereco de z: 000000000062FE44

Pressione qualquer tecla para continuar. . .
```

Quando se define uma variável como ponteiro, dizemos que o endereço de uma variável simples está guardado em um ponteiro que pode ser utilizado como parâmetro para uma função. Para que isso ocorra, basta colocar o operador “*” antes da variável e o operador “&”1 na chamada do parâmetro(Mizrahi, 2008).

O resultado do operador &, "endereço de", sempre será o endereço de memória do elemento em questão, normalmente é o local onde uma variável está alocada na memória. Isto é, esse operador gera um ponteiro.

A função scanf() espera que o usuário digite algum dado de entrada e o operador '&', acompanhado da variável, serve para especificar o lugar certo onde esse dado vai ficar posicionado na memória. Portanto, o uso do operador de endereço para essa função se faz necessário (Ascencio, 2012).

O operador oposto é o * (asterisco), que pega o valor apontado pelo endereço. No exemplo do algoritmo da Figura 1, foram declaradas três variáveis inteiras. Obtivemos como saída a impressão dos seus endereços em hexadecimal com o uso do operador &. O endereço alocado depende de vários fatores, dentre eles, o tamanho da palavra2, se há ou não outros programas usando a memória, entre outros.

Por essas razões, podemos encontrar endereços diferentes na passagem de parâmetros e execução do algoritmo (Mizrahi, 2008) para cada nova execução de um problema. Faça o teste você mesmo. Implemente o exemplo anterior no seu compilador e mande-o executar diversas vezes. Cada nova execução gerará valores diferentes de endereços alocados.

Mizrahi (2008) descreve memória como uma unidade organizada logicamente em palavras. Uma palavra é uma unidade lógica de informação constituída por um número de bits de único endereço, consequentemente, um conjunto de palavras armazenadas na memória é um programa, e pode ser dividido em duas categorias:

- Instruções – operações (programa propriamente dito) realizadas pela máquina;
- Dados – variáveis, ou valores, processadas nessas operações.

Cada palavra é identificada por meio de um endereço de memória sem ambiguidade. Observe a Tabela 1.

Tabela 1 – Exemplo de endereços de palavras

Ordem na memória	Endereço na memória	Palavras
0	000	Palavra 0
1	001	Palavra 1
2	010	Palavra 2
3	011	Palavra 3
4	100	Palavra 4
5	101	Palavra 5
6	110	Palavra 6
7	111	Palavra 7

A capacidade, ou tamanho, de uma memória vai depender do número de palavras que ela pode suportar. A posição de uma palavra dentro da memória é tida como o seu endereço. A primeira palavra da memória tem o endereço 000, a próxima, 001, e assim por diante (Mizrahi, 2008).

TEMA 2 – PONTEIROS

O ponteiro é uma ferramenta poderosa oferecida em linguagens de programação e considerada, pela maioria dos programadores, um dos tópicos mais difíceis (Mizrahi, 2008; Ascencio, 2012).

Apontadores, ou ponteiros, são variáveis que armazenam o endereço de outras variáveis na memória. Ou seja, em vez de termos um valor numérico ou caracteres, por exemplo, armazenado na variável, temos um endereço. Dizemos que um ponteiro “aponta” para uma variável na memória quando este contém o endereço daquela variável.

O uso de ponteiros é muito útil quando um dado deve ser acessado na memória em diferentes partes de um programa. Assim, podem existir vários ponteiros espalhados, indicando a localidade da variável que contém o dado desejado. Caso este dado seja atualizado, todas as partes que apontam para a variável serão atualizados simultaneamente (Ascencio, 2012).

De acordo com Mizrahi (2008), estas são algumas razões para o uso de ponteiros:

1. Fornecem maneiras com as quais as funções podem realmente modificar
os argumentos que recebem (passagem de parâmetros por referência);
2. Criar estruturas de dados complexas, como listas encadeadas e árvores
binárias, em que um item deve conter referências a outro;
3. Alocar e desalocar memória dinamicamente do sistema;

4. Passar para uma função o endereço de outra função.
A sintaxe de declaração de um ponteiro é: *tipo *nome_ponteiro;*
Em que temos:

- **tipo** – refere-se ao tipo de dado da variável armazenada que é apontada pelo endereço do ponteiro;
- ***nome_ponteiro** – o nome da variável ponteiro;
- O uso do asterisco * serve para determinar que a variável usada será um ponteiro.

Um ponteiro, como qualquer variável, deve ser tipificado, que é a identificação do tipo da variável para a qual ele aponta. Para declarar um ponteiro, especifica-se o tipo da variável para a qual ele aponta com o nome precedido por asterisco. Exemplo:

- int ponteiro; // declaração de uma variável comum do tipo inteiro
- int *ponteiro; // declaração de um ponteiro para um inteiro

É importante prestar bastante atenção na hora de declarar vários ponteiros em uma linha, visto que o asterisco deve vir antes de cada nome de variável. Exemplos:

- int x, y, z; // Essa instrução declara três variáveis comuns.
- int *x, y, z; // Essa instrução declara somente x como ponteiro.
- int *x, *y, *z; // Essa instrução declara três ponteiros.

Um ponteiro é uma variável que armazena um endereço de memória, a localização de outra variável. Dizemos que uma variável aponta para outra quando a primeira contém o endereço da segunda. Observe os exemplos mostrados nas figuras 3 e 4.

Figura 3 – Exemplo de ponteiros

```c
#include<stdio.h>
#include<stdlib.h>

int  main ()
{
    int *y, x;


    printf( "Digite um numero: ");
    scanf("%d", &x);

    y = &x; // y recebe o endereço de x

    // Imprime o dado que y aponta
    printf("\n Voce digitou o numero: %d \n", y);

    system ("pause");
    return  0;
}
```

A Figura 4 mostra a saída do algoritmo acima após a sua execução.

Figura 4 – Saída do algoritmo

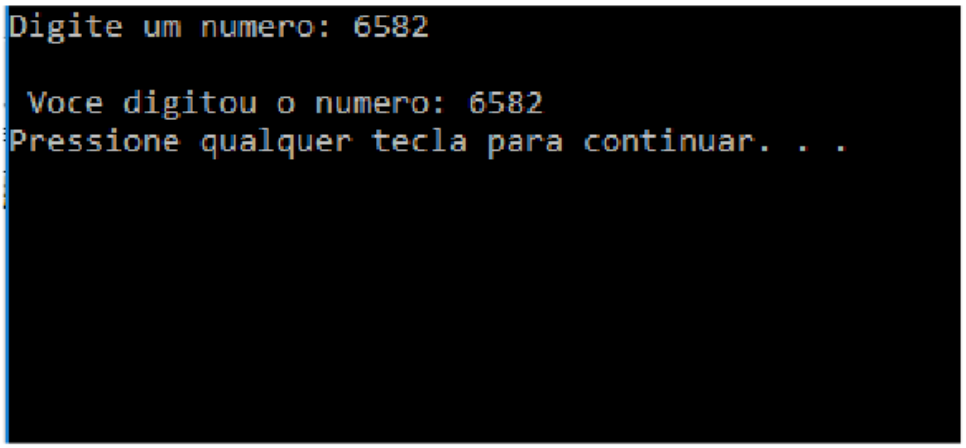

Figura 5 – Exemplo de ponteiros

```c
#include<stdio.h>
#include<stdlib.h>

int  main()
{
    int x = 4 , y = 7 ; //varaveis do tipo inteiro x e y
    int *px, *py;       //ponteiros do tipo inteiro px e py

    //imprime os enderecos e os dados das variaveis x e y
    printf ("Endereco (&x) = %p --- Dado (x) = %d \n" , &x, x);
    printf ("Endereco (&y) = %p --- Dado (y) = %d \n\n" , &y, y);

    px = &x; //px recebe o endereço de x
    py = &y; //py recebe o endereço de y

    /*imprime os enderecos apontados e os dados
    das variaveis referenciadas */
    printf ("Endereco (px) = %p --- Dado (*px) = %d \n" , px, *px);
    printf ("Endereco (py) = %p --- Dado (*py) = %d \n\n" , py, *py);

    system ("pause");
    return  0 ;
}
```