Ponteiros

Prof. Daniel Di Domenico
ddomenico@inf.ufsm.br

Ponteiros

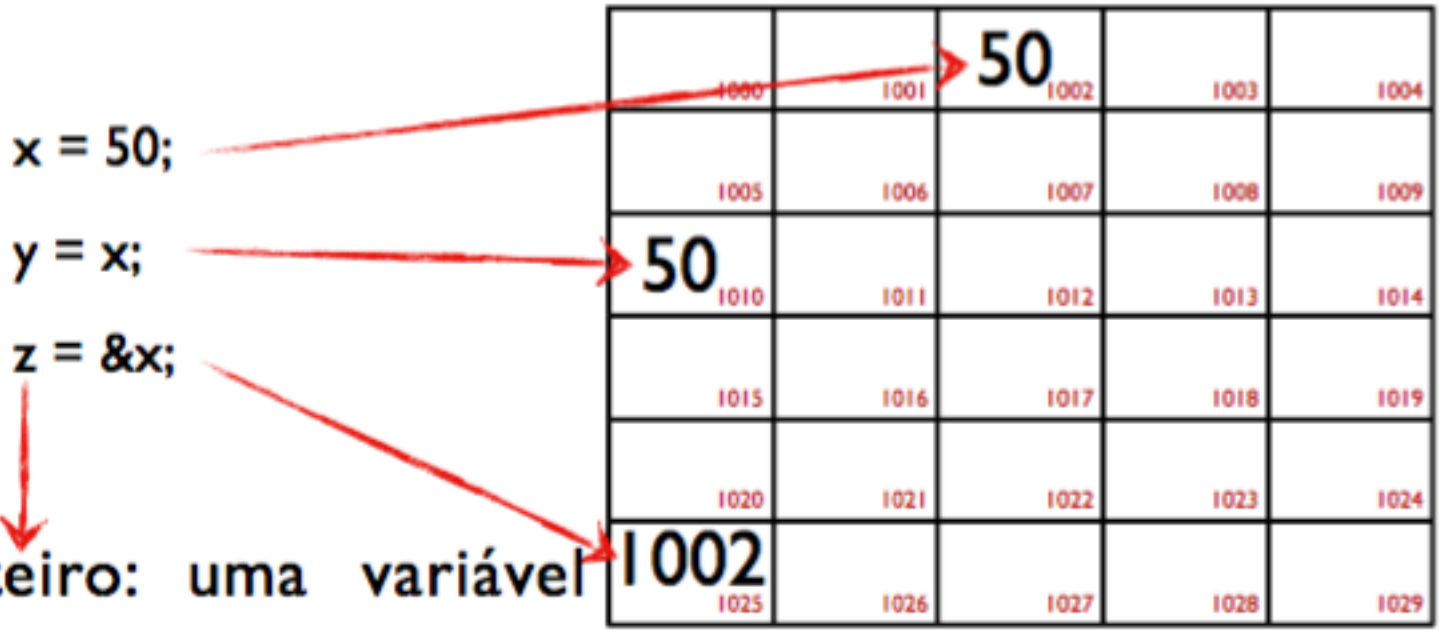
(apontadores, referências)

- Quando declaramos uma variável, uma quantidade de memória é alocada em algum lugar específico (endereço de memória);
- O programador não decide em qual endereço uma variável será armazenada;
- Por vezes, pode ser necessário conhecer o endereço de memória de uma variável;
- O endereço de memória de uma variável é uma referência para a variável;

Ponteiros

(apontadores, referências)

- O endereço de uma variável pode ser obtido usando o operador de referência, &, antes do nome da variável;

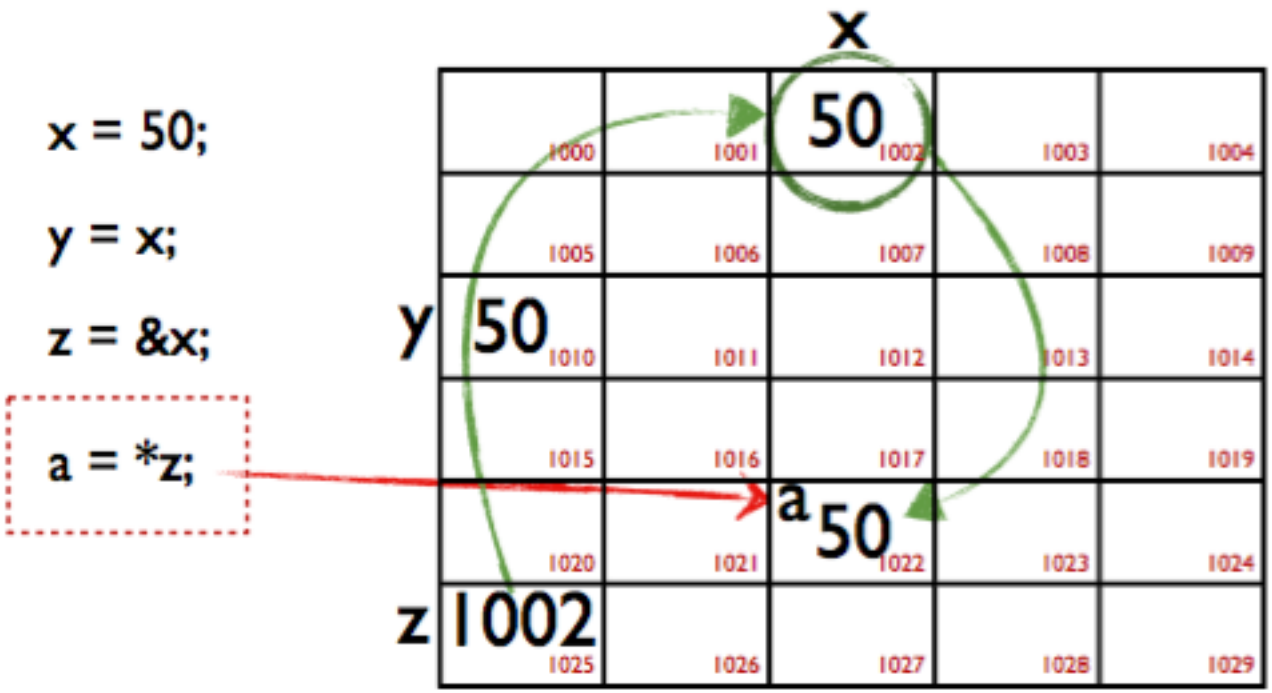


Ponteiro: uma variável que armazena uma referência para outra variável.

Ponteiros

(apontadores, referências)

- Usando um ponteiro podemos obter o valor armazenado na variável que é referenciada (apontada);
- Para tal, devemos preceder o ponteiro com o operador de dereferência, *, (valor apontado por...);



Ponteiros

(apontadores, referências)

- Qual é a diferença entre z e *z?

```
a = z;
a = *z;
```

a recebe o valor de z (1002)

a recebe o valor apontado por z (50)

Ponteiros

(declaração de variáveis do tipo ponteiro)

- A forma geral da declaração de ponteiros é:

```
tipo * nome;

int * v;
float * vmedio;
char * letra;
```

- Ponteiros com tipos diferentes usam a mesma quantidade de memória;

Tamanho de Variáveis X Tamanho de Ponteiros

TIPO	TAMANHO BYTES
CHAR	1
INT	4
FLOAT	4
DOUBLE	8

- Depende de alguns fatores:
  - Arquitetura da CPU (tamanho da palavra da arquitetura do processador);
  - Sistema Operacional.
    - SO 32 bits = 4 bytes;
    - SO 64 bits = 8 bytes.

Ponteiros
(exemplo 1)

```
//exe1Ponteiros.c

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

int main() {
    int valor1, valor2;
    int *ponteiro;

    ponteiro = &valor1;
    *ponteiro = 10;

    ponteiro = &valor2;
    *ponteiro = 20;

    printf("Valor 1 é %d.\n", valor1);
    printf("Valor 2 é %d.\n", valor2);
}
```

Ponteiros

(exemplo 2 – recebendo diversos valores ao longo do programa)

```
//exe2Ponteiros.c

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

int main() {
    int valor1 = 5, valor2 = 15;
    int *p1, *p2;

    p1 = &valor1; //p1 = endereço de valor1
    p2 = &valor2; //p2 = endereço de valor2
    *p1 = 10;     //valor do endereço apontado por p1 = 10;
    *p2 = *p1;    //valor apontado por p2 = valor apontado por p1
    p1 = p2;      //p1 = p2 (valor do ponteiro é copiado)
    *p1 = 20;     //valor apontado por p1 = 20

    printf("Valor 1 é %d.\n", valor1);
    printf("Valor 2 é %d.\n", valor2);

    return 0;
}
```

Ponteiros

(e vetores)

- O identificador de um vetor equivale ao endereço do seu primeiro elemento;
- Um ponteiro equivale ao endereço do primeiro elemento para o qual aponta; (mesmo conceito?)

```
int vnum[10];
int * p;
p = vnum;
```

p e vnum são equivalentes. Valor de p pode ser alterado. Valor de vnum sempre apontará para o primeiro elemento do vetor (ponteiro constante).

```
vnum = p;
```
declaração inválida!

Ponteiros
(exemplo 3)

```
//exe3Ponteiros.c

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

int main() {
    int i, vnum[5];
    int *p1;
    p1 = vnum; *p1 = 10;
    p1++; *p1 = 20;
    p1 = &vnum[2]; *p1 = 30;
    p1 = vnum + 3; *p1 = 40;
    p1 = vnum; *(p1+4) = 50;

    for(i=0; i < 5; i++)
        printf("%d\t", vnum[i]);
    printf("\n");

    return 0;
}
```

Ponteiros

- Situações encontradas em ponteiros que acabam confundindo em algumas situações:

  int *p = determinado endereço de memória;

  - p++: incrementa o ponteiro, ou seja o endereço. Após esta instrução, o ponteiro **p** passará a apontar para a posição de memória imediatamente seguinte.
  - (*p)++: Incrementa o conteúdo apontado por p, ou seja, o valor armazenado na variável para qual **p** está apontando.
  - *(p++): Incrementa **p** (como em p++) e acessa o valor encontrado na nova posição. Se em um vetor, esta expressão acessa o valor da posição imediatamente superior a armazenada em **p** antes do incremento.

Ponteiros
(exemplo 4)

```
//exe4Ponteiros.c

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

int main() {
    int i, vnum[5];
    int *p1;
    for(i=0; i < 5; i++)
        vnum[i] = (i+1) * 10;

    p1 = vnum; p1++;
    printf("%d\n", *p1);

    p1 = vnum; (*p1)++;
    printf("%d\n", *p1);

    p1 = &vnum[2]; int aux = *(p1++);
    printf("%d %d\n", aux, *p1);

    return 0;
}
```

Ponteiros

(e vetores)

- Em vetores (ou matrizes) usamos colchetes, [ ], para especificar o índice de um elemento;
  - Colchetes são operadores de dereferência;
  - Eles dereferenciam a variável da mesma forma que * faz em ponteiros, mas indicam o endereço na estrutura a ser dereferenciada;

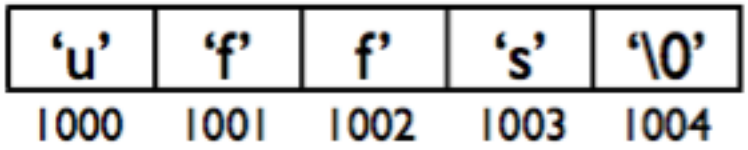
```
vnum[3] = 0;
*(vnum+3) = 0;
```

Ponteiros

- É possível inicializar um ponteiro com o valor para o qual aponta;

```
char * p1 = "uffs";
```

- Neste caso é reservado um espaço de memória para armazenar "uffs" e o endereço do primeiro elemento deste bloco de memória é atribuído ao ponteiro p1;



p1 1000

- O ponteiro p1 aponta para uma sequência de caracteres e pode ser lido como um vetor;
  - ex.: `*(p1+3);` ou `p1[3];` para obter a letra 's'.

Ponteiros **void ***

(exemplo 5 – ponteiros que podem ser utilizados para qualquer tipo de dado)

```
//exe5Ponteiros.c

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

int main() {
    int v1 = 10;
    float v2 = 2.50;

    void *p; //ponteiro genérico

    p = &v1; //p aponta para um inteiro
    printf("%d\n", *(int *)p);

    p = &v2; //p aponta para um float
    printf("%0.2f\n", *(float *)p);

    return 0;
}
```

Ponteiros NULL

- Ponteiro de qualquer tipo que tem um valor especial que indica que não aponta para qualquer endereço válido na memória;

```
int * p;
p = NULL;
```

p é um ponteiro de valor nulo

Ponteiros

(exemplo 6 – passagem de ponteiros por parâmetro)

```
//exe6Ponteiros.c

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

void troca (int *p, int *q);

int main() {
    int v1 = 10, v2 = 20;
    troca(&v1, &v2);

    printf("Valor 1: %d\n", v1);
    printf("Valor 2: %d\n", v2);
    return 0;
}

void troca (int *p, int *q) {
   int temp;
   temp = *p; *p = *q; *q = temp;
}
```