Laboratório de programação

Ponteiros

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

- Um **ponteiro**, é uma variável que guarda um endereço de memória.
- Declaração:

```
int *contPtr, cont;
```

- Quando o * é usado em uma declaração, ele indica que a variável declarada é um ponteiro.

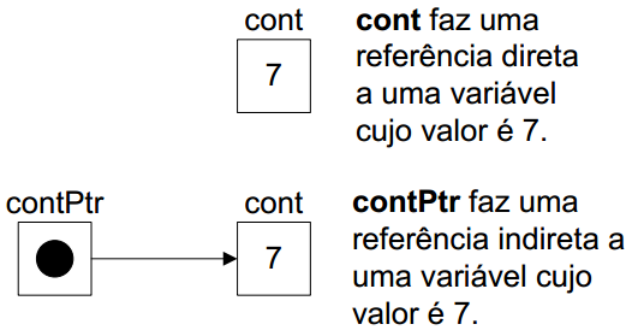

Figura: Fazendo referência direta e indireta a uma variável.

- Os ponteiros devem ser inicializados ao serem declarados ou em uma instrução de atribuição.
- Um ponteiro pode ser inicializado com 0, NULL ou um endereço.
- Um ponteiro com o valor NULL não aponta para lugar algum.
- NULL é uma constante simbólica definida no arquivo de cabeçalho `<stdio.h>`.
- O valor 0 é o único valor inteiro que pode ser atribuído diretamente a uma variável de ponteiro.

- O & ou operador de ponteiro é um operador unário que retorna o endereço de seu operando. Por exemplo, admitindo as declarações

```
1 int y = 5;
2 int *yPtr;
```

a instrução

```
1 yPtr = &y;
```

atribui o endereço da variável **y** à variável de ponteiro **yPtr**. Diz-se que a variável **yPtr** "aponta para" **y**.

- O operador *, chamado frequentemente **operador de referência indireta** ou **operador de desreferenciamento**, retorna o valor do objeto ao qual seu operando (i.e., um ponteiro) aponta. Por exemplo, a instrução

```
printf("%d", *yPtr);
```

imprime o valor da variável **y**, ou seja, 5. Usar * dessa maneira é chamado *desreferenciar* um poteiro.

Warning!

Desreferenciar um ponteiro que não foi devidamente inicializado ou que não foi atribuído para apontar para um local específico da memória. Isso poderia causar um erro fatal de tempo de execução, ou poderia modificar acidentalmente dados importantes e permitir que o programa seja executado até o final fornecendo resultados incorretos.

```c
#include <stdio.h>
int main(){
    int a; /* a e um inteiro */
    int *aPtr; /* aPtr e um ponteiro para um inteiro */
    a = 7;
    aPtr = &a; /* aPtr define o endereço de a */

    printf("O endereço de a e %p\n"
        "O valor de aPtr e %p\n\n", &a, aPtr);

    printf("O valor de a e %d\n"
        "O valor de *aPtr e %d\n\n", a, *aPtr);

    printf("Sabendo que * e & complementam-se mutuamente."
        "\n&*aPtr = %p\n*&aPtr = %p\n",&*aPtr, *&aPtr);

    return 0;
}
```

Exercício 31

Crie um programa para calcular a área e o perímetro de um hexágono. O seu programa deve implementar uma função chamada **calculaHexagono** que calcule a área e o perímetro de um hexágono regular de lado **L**. A função deve obedecer o seguinte protótipo: `void calculaHexagono(float L, float *area, float *perimetro);`
OBS: área do hexágono

$$A = \frac{3L^2\sqrt{3}}{2} \quad P = 6L$$

- Um conjunto limitado de operações aritméticas pode ser realizado com ponteiros. Um ponteiro pode ser
  - incrementado (++) ou decrementado (−−),
  - adicionado um inteiro a um ponteiro (+ ou +=),
  - um inteiro pode ser subtraído de um ponteiro (- ou - =) ou
  - um ponteiro pode ser subtraído de outro.

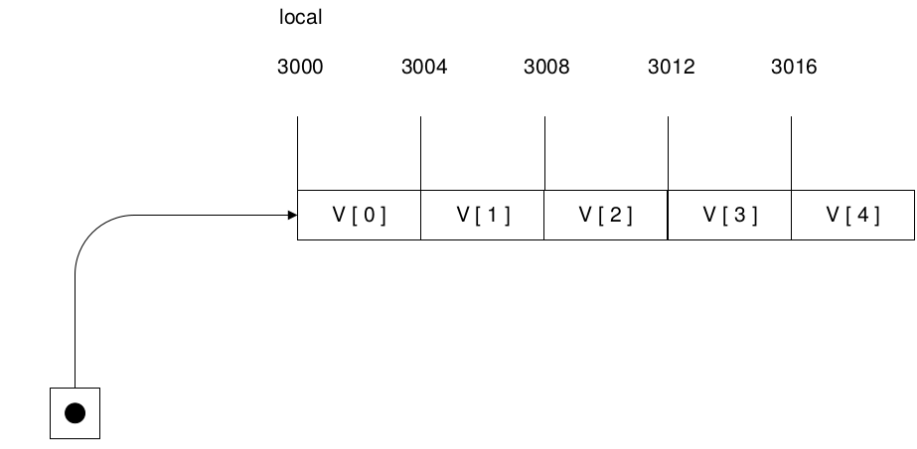

Figura: O array **v** e uma variável de ponteiro **vPtr** que aponta para **v**.

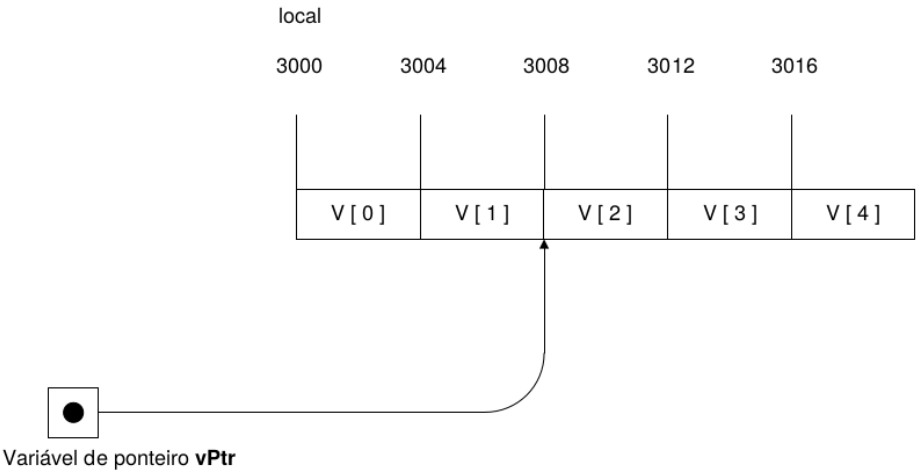

Figura: O ponteiro **vPtr** após a aritmética de ponteiros (vPtr += 2).

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

int main(){
    int v[9] = {1,2,3,4,5,6,7,8,9},i;
    int *ptr = NULL;

    ptr = v;

    printf("*ptr\tptr\n");
    for (i = 0; i < 9; i++){
        printf("%2d\t%p\n", *ptr, ptr);
        ptr++;
    }
}
```

- A linguagem C fornece o operador unário especial **sizeof** para determinar o tamanho em bytes de qualquer tipo de dado.
- O número de elementos em um array também pode ser determinado em tempo de compilação.

```
double vet[22];
```

  Normalmente, as variáveis do tipo double são armazenadas em 8 bytes de memória. Assim, o array real contém um total de 176 bytes.
- Para determinar o número de elementos do array, pode ser usada a expressão a seguir:

```
sizeof(vet) / sizeof(double)
```

```c
#include <stdio.h>
int main(){
    printf("%-12s\t%15s\n", "TIPO", "TAMANHO (bytes)");
    printf("%-12s\t%2d\n", "char", sizeof(char));
    printf("%-12s\t%2d\n", "short", sizeof(short));
    printf("%-12s\t%2d\n", "int", sizeof(int));
    printf("%-12s\t%2d\n", "long", sizeof(long));
    printf("%-12s\t%2d\n", "float", sizeof(float));
    printf("%-12s\t%2d\n", "double", sizeof(double));
    printf("%-12s\t%2d\n", "long double", sizeof(long double));
    return 0;
}
```

- Os arrays e os ponteiros estão intimamente relacionados em C e um ou outro podem ser usados quase indiferentemente.
- Pode-se imaginar que o nome de um array é um ponteiro constante.
- Admita que o array inteiro **b[5]** e a variável de ponteiro **bPtr** foram declarados. Como o nome do array (sem um índice) é um ponteiro para o primeiro elemento do array.

```
bPtr = b;
```

Essa instrução é equivalente a tomar o endereço do primeiro elemento do array

```
bPtr = &b[0];
```

- O elemento **b[3]** do array pode ser referenciado alternativamente com a expressão de ponteiro

```
*(bPtr + 3)
```

O 3 na expressão anterior é o *offset* (deslocamento) do ponteiro.

```c
#include <stdio.h>

int main(){
    int b[] = {10, 20, 30, 40};
    int *p;

    p = b;
    printf("%d\n", *(p + 2));

    return 0;
}
```

Exercício 32

Crie uma função que receba como parâmetro um vetor e o imprima. Não utilize índices para percorrer o vetor, apenas aritmética de ponteiros.

- Os arrays podem conter ponteiros. Um uso comum de tais estruturas de dados é para formar um array de strings.

```
char *naipe[4] = {"Copas", "Ouros", "Paus", "Espadas"};
```

- Cada elemento do array é uma string.
- Mas em C uma string é essencialmente um ponteiro para seu primeiro caractere.
- Assim, cada elemento de um array de strings é na verdade ura ponteiro para o primeiro caractere de uma string.

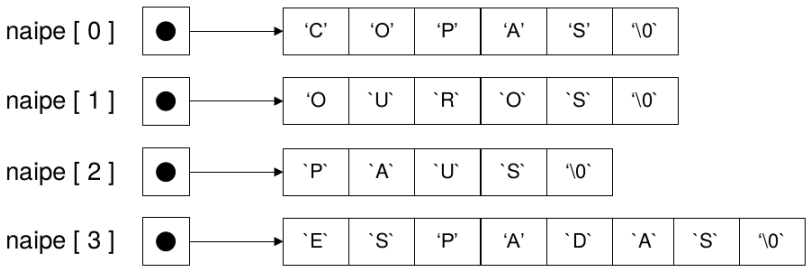

Figura: Um exemplo gráfico do array naipe.

- Um ponteiro para uma função é uma variável que pode conter o endereço de uma função na memória.
- Vimos que o nome de um array representa o endereço do primeiro elemento do array na memória. Similarmente, o nome de uma função é na realidade uma representação do endereço onde o código da função na memória.
- Os ponteiros para funções podem ser passados a funções, retornados de funções, armazenamos em arrays e atribuídos a outros ponteiros de funções.

- Um ponteiro para função funciona de forma semelhante a um ponteiro para variável.
- Porém ele armazena o endereço de uma função.
- Esta é a sintaxe de definição de um ponteiro para função:

`tipo_retorno (*nome_ponteiro)(lista de parâmetros)`

```
1 int *p;               //ponteiro para inteiro
2 int (*f)(int, int); //pont. para funcao: retorno e parametros inteiros
```

```c
#include <stdio.h>
void portugues() {
    printf("Olá mundo!\n");
}
void ingles() {
    printf("Hello world!\n");
}
int main() {
    void (*p)();
    p = portugues;
    p();
    p = ingles;
    p();
    return 0;
}
```

```c
#include <stdio.h>

int quadrado(int x) {
    return x*x;
}

int main() {
    int (*p)(int);

    p = quadrado;

    printf("5**2 = %d\n", p(5));
    return 0;
}
```

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

struct Heroi {
    char nome[20];
    int forca;
    void (*habilidade)(Heroi *);
};

void raioLaser(struct Heroi *he) {
    printf("Ziiiimmmmm!\n");
    he->forca = he->forca - 5;
}

void soco(struct Heroi *he) {
    printf("Poooowww!\n");
    he->forca = he->forca - 100;
}
```

```
20  int main() {
21      struct Heroi h1 = {"Superman", 100, raioLaser};
22      struct Heroi h2 = {"Saitama", 100, soco};
23
24      h1.habilidade(&h2);
25      h2.habilidade(&h1);
26
27      printf("%s (força = %d)\n", h1.nome, h1.forca);
28      printf("%s (força = %d)\n", h2.nome, h2.forca);
29      return 0;
30  }
```

-

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

int soma(int a, int b){ return a+b; }
int subtrai(int a, int b){ return a-b; }
int multiplica(int a, int b){ return a*b; }
int divide(int a, int b){ return a/b; }
int calc(int a, int b, int (*op)(int,int)){
    return op(a, b);
}

int main() {
    int a, b, opcao;
    int (*op[4])(int,int) = {soma, subtrai, multiplica, divide};

    printf("Digite dois numeros:\n");
```

```
17      scanf("%d%d",&a,&b);
18
19      printf("Operacao:\n");
20      printf(" 0 soma\n 1 subtracao\n 2 multiplicacao\n 3 divsao\n:");
21      scanf("%d",&opcao);
22
23      if (opcao >= 0 && opcao < 4)
24          printf("Resultado = %d\n", calc(a, b, op[opcao]));
25      else
26          printf("Opção inválida.\n");
27
28      return 0;
29  }
```

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

int comparador(const void *valor1, const void *valor2) {
    int *a = (int*) valor1;
    int *b = (int*) valor2;
    return *a - *b;
}

int main() {
    int i, vetor[15] = {2,5,11,9,10, 4,3,13,1,15, 7,6,8,14,12};

    qsort(vetor, 15, sizeof(int), comparador);

    for (i=0; i<15; i++)
        printf("%d ", vetor[i]);
    printf("\n");
    return 0;
}
```