

Linguagem de Programação I

- Ponteiros - Parte 1 -

Prof. Ulysses Santos Sousa
ulyssessousa@ifma.edu.br

Aula 08

Roteiro

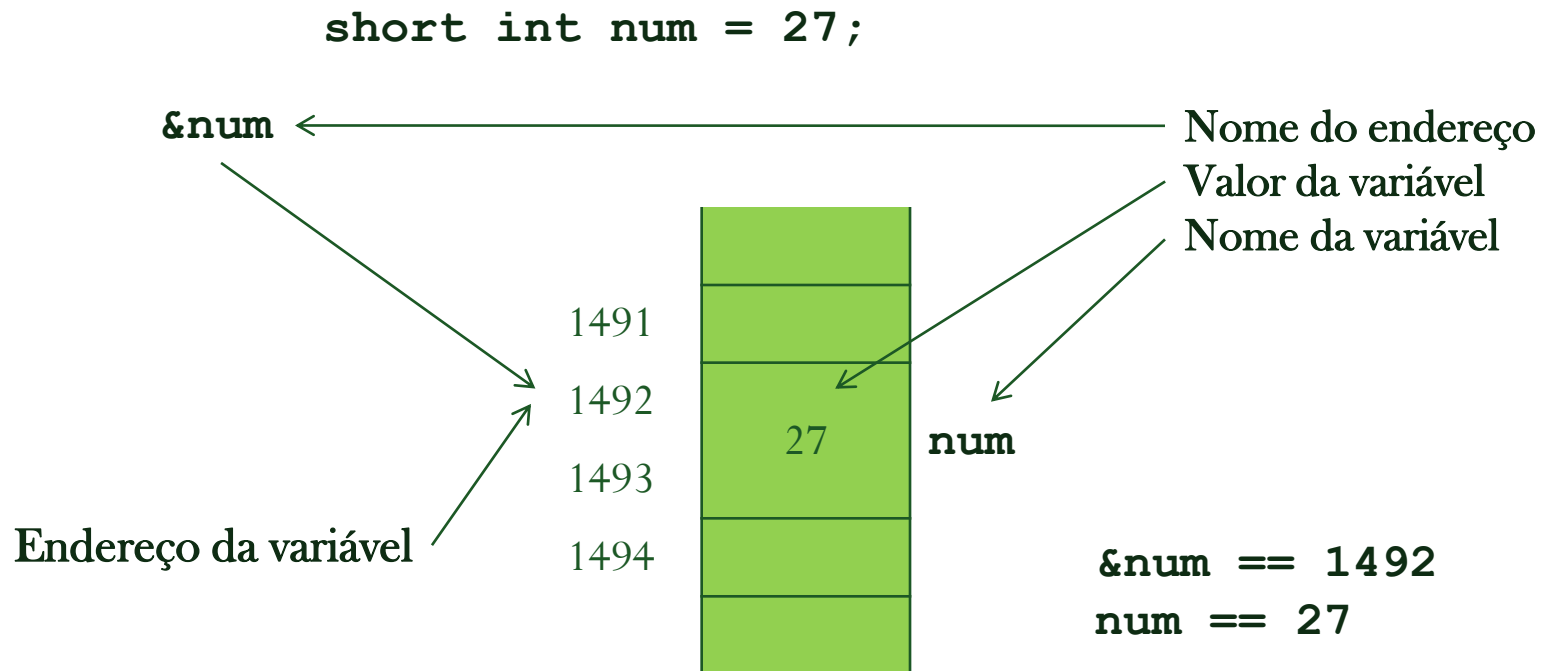
- Endereços de memória
- Endereços de variáveis
- Definição de ponteiros
- Por que utilizar ponteiros?
- Ponteiros
 - Tipos, declaração, operador de endereço, operador indireto.
- Operações com ponteiros
- Ponteiros e vetores
- Ponteiros constantes e ponteiros variáveis
- Passagem de vetores como argumento para funções

Endereços de memória

- A memória é dividida em bytes, numerados de zero até o limite de memória da máquina.
- Esses números são chamados de *endereços de bytes*.
- O endereço de uma variável na memória é o primeiro byte ocupado por ela.

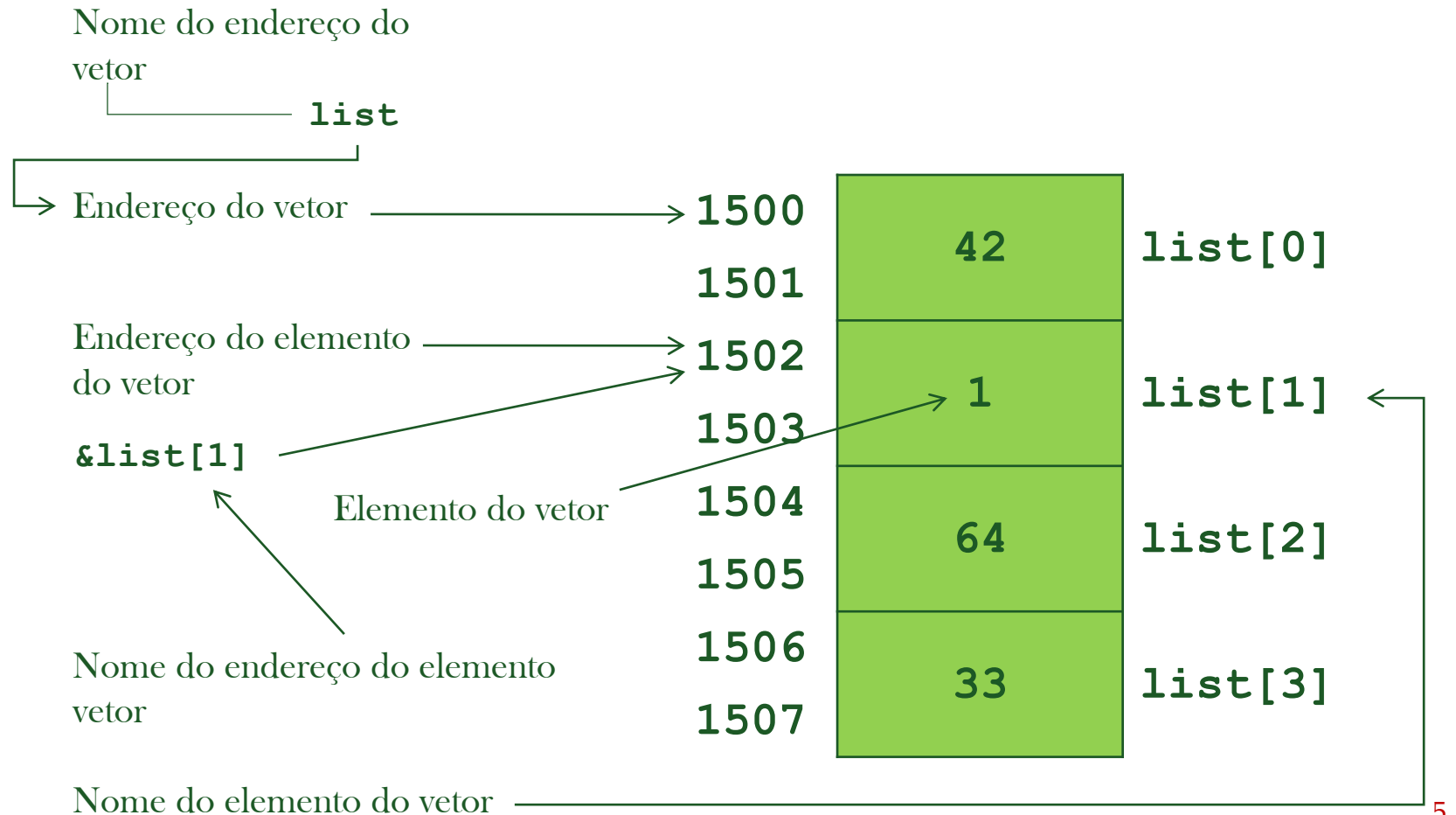
Endereços de variáveis

- Variável simples



Endereços de variáveis

- Vetores



Definição de ponteiro

- **Ponteiro**

- É um tipo especial de variável que armazena endereços.
- O valor do ponteiro indica em que parte da memória uma variável está alocada;
- Proporcionam um modo de acesso à variável **sem** referenciá-la diretamente.

Um ponteiro pode ter um valor especial, **NULL**, que não representa nenhum endereço.



A constante NULL está definida na biblioteca stdlib.

Por que utilizar ponteiros?

- **Porque eles possibilitam:**
 - Que funções alterem o valor dos parâmetros que recebem (passagem por referência);
 - Passar matrizes e strings de forma mais adequada de uma função para outra;
 - Manipulação dos elementos de matrizes fica mais fácil, por meio da movimentação de ponteiros para elas;
 - Criação de estruturas de dados complexas, em que um item pode ter referência a outro.
 - Alocar e desalocar memória dinamicamente do sistema;
 - Passar para uma função o endereço de outra função.

Ponteiros

- **Comparação entre ponteiros e “variáveis normais”**
 - Uma "variável normal" é um local na memória que pode conter um valor.
 - Um **ponteiro** é uma variável que **aponta** para outra variável. Isto significa que um ponteiro mantém o endereço de memória de outra variável.
 - O ponteiro não contém um valor no sentido tradicional, mas sim o endereço de outra variável. Um ponteiro "aponta para" esta outra variável mantendo uma cópia de seu endereço.

Ponteiros

- **Tipos de ponteiro**

- No C quando declaramos ponteiros nós informamos ao compilador para que tipo de variável vamos apontá-lo.
- O tipo base do ponteiro define que tipo de variáveis que o ponteiro pode apontar.
- Toda a aritmética de ponteiros é feita por meio do tipo base.

Ponteiros

- Declaração de um ponteiro

- Forma geral:

```
tipo_do_ponteiro *nome_da_variável;
```

É o asterisco (*) que faz o compilador saber que aquela variável não vai guardar um valor mas sim um endereço para aquele tipo especificado.

- Exemplo:

```
int *pt;
```

→ *declara um ponteiro para int*

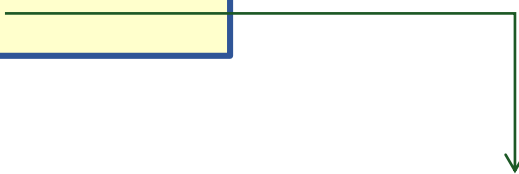
```
char *temp, *pt2;
```

→ *declara dois ponteiros para caractere*

Ponteiros

- **Declaração e utilização de uma variável ponteiro**
 - Um ponteiro deve ser inicializado antes de ser usado;
 - Para atribuir um valor a um ponteiro recém-criado poderíamos igualá-lo a um valor de memória.
 - **Exemplo:**

```
int count = 10;  
int *pt;  
pt = &count;
```

A horizontal line extends from the end of the third line of code ('pt = &count;') to the right. From the end of this line, a vertical arrow points downwards.

Para saber o endereço de uma variável
basta usar o operador &.

Ponteiros

- **Operador de Endereços (&)**

- O & é um operador que devolve o endereço de memória do seu operando:

```
int *m;  
m = &count;
```

O operador & pode ser encarado como retornando o “endereço de”.

Ponteiros

- **Operador Indireto (*)**

- O * é um operador unário que devolve o valor da variável localizada no endereço que o segue.
- É complementar de &.

```
m = &count;  
q = *m;
```



O operador * pode ser imaginado como “no endereço”.

Ponteiros

- Exemplo:

```
#include <stdio.h>
int main (){
    int num,valor;
    int *p;
    num=55;

    p=&num;          // Pega o endereco de num
    valor=*p;        // Valor e igualado a num de uma
                    // maneira indireta

    printf("\n\n%d\n", valor);
    printf("End. p onde o ponteiro aponta: %p\n",p);
    printf("Valor da variavel apontada: %d\n",*p);
    return 0;
}
```

Ponteiros

- Exemplo:

```
#include <stdio.h>
int main (){
    int num,*p;
    num=55;

    p=&num;        // Pega o endereco de num

    printf("\nValor inicial: %d\n",num) ;

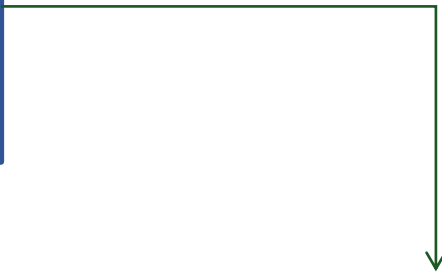
    *p=100;        //Muda o valor de num de uma maneira
                  //indireta
    printf("\nValor final: %d\n",num) ;
    return 0;
}
```

Operações com ponteiros

- **Atribuição**

- Um ponteiro pode receber o valor de outro ponteiro, desde que os ponteiros sejam do mesmo tipo.

```
...  
float *p1, *p2, val;  
p1 = &val;  
p2 = p1;  
...
```



p1 recebe o endereço de val e p2
recebe o conteúdo de p1 (endereço
de val)

Operações com ponteiros

- **Aritmética de Ponteiros**

- Uma quantidade inteira pode ser adicionada ou subtraída de um ponteiro.
- A adição de um inteiro n a um ponteiro p fará com que ele aponte para o endereço do n -ésimo bloco seguinte.

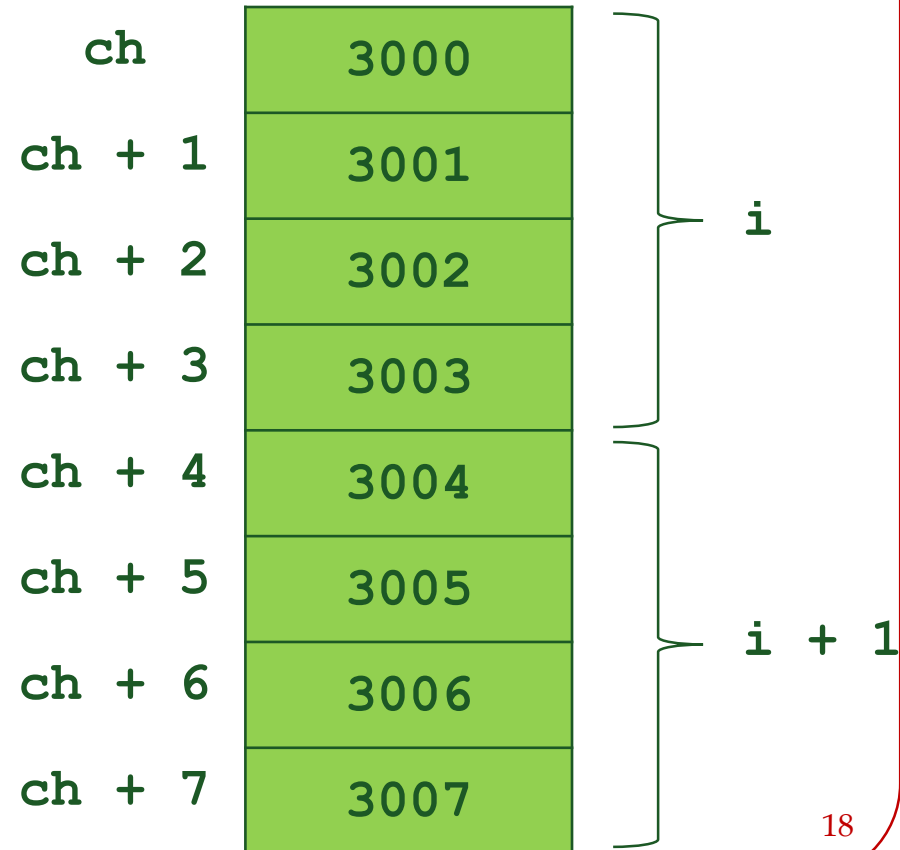
```
...  
double *p, *q, var;  
p = &var  
q = ++p;    //q aponta para o bloco seguinte ao  
            //ocupado por var  
p = q - 5;  // p aponta para o quinto bloco  
            //anterior a q  
...
```

Operações com ponteiros

- Toda aritmética de ponteiros é relativa a seu tipo base

```
char *ch=3000;  
int *i=3000;
```

✓ A unidade com ponteiros é o número de bytes do tipo apontado.



Operações com ponteiros

- Para incrementar o conteúdo da variável apontada pelo ponteiro p , faz-se:

```
(*p) ++ ;
```

- E se você quiser usar o conteúdo do ponteiro 15 posições adiante:

```
* (p+15) ;
```

Operações com ponteiros

- **Comparação de Ponteiros**

- Dois ponteiros podem ser comparados (usando-se operadores lógicos) desde que sejam de mesmo tipo.

```
...  
if(px == py) {      //se px aponta para o mesmo bloco que py  
if(px > py) {        //se px aponta para um bloco posterior a py  
if(px != py) {      //se px aponta para um bloco diferente de py  
if(px == NULL) {    // se px é nulo  
...  

```

Ponteiros e vetores

- Em C, o compilador transforma vetores e matrizes em ponteiros, pois a arquitetura do microcomputador compreende ponteiros, e não matrizes.
- Qualquer operação feita com índices de um vetor pode ser feita com ponteiros.
- O nome de um vetor é um endereço, ou seja, um ponteiro.

Ponteiros e vetores

- Exemplo com vetores:

```
main() {  
    int nums[]={92, 81, 70, 69, 58};  
    int d;  
    for(d=0; d<5; d++){  
        printf("%d\n", nums[d]);  
    }  
}
```

- O mesmo exemplo com ponteiros:

```
main() {  
    int nums[]={92, 81, 70, 69, 58};  
    int d;  
    for(d=0; d<5; d++){  
        printf("%d\n", *(nums+d));  
    }  
}
```

Ponteiros e vetores

- As duas versões dos programas anteriores são idênticas, exceto a expressão $*(\text{num} + d)$

$*(\text{vetor} + \text{indice})$ É O MESMO QUE $\text{vetor}[\text{indice}]$

- Se o endereço do vetor é 3000:

```
&nums[2] == (nums + 2) == 3004  
nums[2] == *(nums + 2) == 70
```

Ponteiros constantes e ponteiros variáveis

- **Ponteiro constante**

- É um endereço, uma simples referência.
- O nome de um vetor é um ponteiro constante.

- **Ponteiro variável**

- É um lugar da memória que armazena um endereço.

Ponteiros constantes e ponteiros variáveis

```
main() {  
    int nums[]={92, 81, 70, 69, 58};  
    int d;  
    for(d=0; d<5; d++){  
        printf("%d\n", *(nums++)); /* ERRO */  
    }  
}
```

```
main() {  
    int nums[]={92, 81, 70, 69, 58};  
    int d, *p = nums;  
    for(d=0; d<5; d++){  
        printf("%d\n", *(p++)); /*SOLUÇÃO COM PONTEIROS*/  
    }  
}
```

Ponteiros constantes e ponteiros variáveis

- Exemplo de ponteiro constante:

```
#define LIM 40
main() {
    float notas[LIM], soma = 0.0;
    int i = 0;
    do{
        printf("Digite a nota do aluno %d:", i);
        scanf("%f", notas+i);
        if(*(notas+i)>0){
            soma += *(notas+i);
        }
    }while(*(notas+i++)>0);
    printf("Media das notas: %.2f", soma/(i));
}
```

Ponteiros constantes e ponteiros variáveis

- Exemplo de ponteiro variável:

```
#define LIM 40
main() {
    float notas[LIM], soma = 0.0, *ptr;
    int i = 0;
    ptr = notas;
    do{
        printf("Digite a nota do aluno %d:", i);
        scanf("%f", ptr);
        if(*ptr > 0){
            soma += *ptr;
        }
    }while(* (ptr++) > 0);
    printf("Media das notas: %.2f", soma/(i));
}
```

Passagem de vetores como argumento para funções

- Podemos utilizar ponteiros variáveis na declaração de uma função que receberá um vetor como parâmetro.
- Protótipo com ponteiro constante:
 - `float media(float [], int);`
- Protótipo com ponteiro variável:
 - `float media(float *, int);`

Passagem de vetores como argumento para funções

- Exemplo:

```
...  
...  
float media(float *, int);  
...  
...  
int main() {  
    float notas[TAMANHO], m;  
    ...  
    m = media(notas, i);  
    ...  
}  
...  
float media(float *lista, int tamanho) {  
    int i;  
    float m = 0.0;  
    for(i = 0; i < tamanho; i++)  
        m += *(lista++);  
    return m/tamanho;  
}
```

Referências

- MIZRAHI, V. V. Treinamento em Linguagem C. 2ª Edição. São Paulo: Person Prentice Hall, 2008.