

DISCIPLINA: ESTRUTURAS DE DADOS

PROGRAMA DE ARTICULAÇÃO DA FORMAÇÃO
PROFISSIONAL MÉDIA E SUPERIOR (AMS)

PONTEIROS
FUNÇÕES

- Prof. Alexandre Ponce de Oliveira
alexandreponceo@gmail.com

Aulas 09 e 10

Lins/SP

Ponteiro

Recurso poderoso da linguagem, também chamados apontadores.

É uma variável que contém o endereço de uma posição de memória.

Microprocessador acessa a memória através desses endereços.

Por exemplo, um microcomputador com 64 MBytes de memória tem suas posições de armazenamento endereçadas de 0 até 67.108.863 (notação decimal) ou de 0 até 3FFFFFFF (notação hexadecimal).

Ponteiro

Variáveis do tipo ponteiro aponta para uma outra variável.

Quando se declara uma variável `i` (`int i;`), por exemplo, e a utiliza em um comando como:

```
i = i + 1;
```

O nome `i` representa o valor que é associado a ela, e não o endereço onde seu valor é armazenado.

Porém, existem casos em que é necessário utilizar o endereço de uma variável e não o seu conteúdo.

Ponteiro

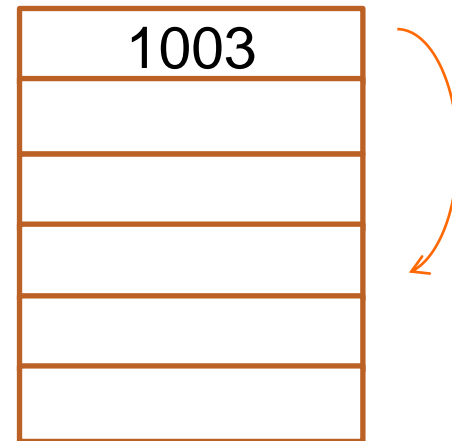
Sintaxe para declara uma variável ponteiro:

Tipo *nome;

Endereço na
memória

1000
1001
1002
1003
1004
1005

Variável na
memória



Ponteiro

Operador &

Devolve o endereço de memória de uma variável, representado pelo caractere & ("e" comercial).

Esse é um dos operadores unários do C, ou seja, ele requer apenas um operando.

O operador +, por exemplo, é considerado binário, pois necessita dois operandos.

O operador & é utilizado à direita do nome da variável da qual se deseja o endereço: &operando.

Para melhor entendimento entre variável e endereço da variável, digite o programa exemplo a seguir:

Ponteiro – Exemplo1

```
#include <stdio.h>
#include <conio.h>
main()
{
    int i,j;
    i = 5;
    j = 10;
    printf("O valor de i = %d\n", i);
    printf("O endereco de &i = %d\n", &i);
    printf("O valor de j = %d\n", j);
    printf("O endereco de &j = %d\n", &j);
    getch();
}
```



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Documents and Settings\Administrator\Meus documentos". The command prompt displays the output of the C program, showing the values of variables i and j, and their memory addresses.

```
C:\Documents and Settings\Administrator\Meus documentos
O valor de i = 5
O endereco de &i = 2293620
O valor de j = 10
O endereco de &j = 2293616
```

Ponteiro

Operador *

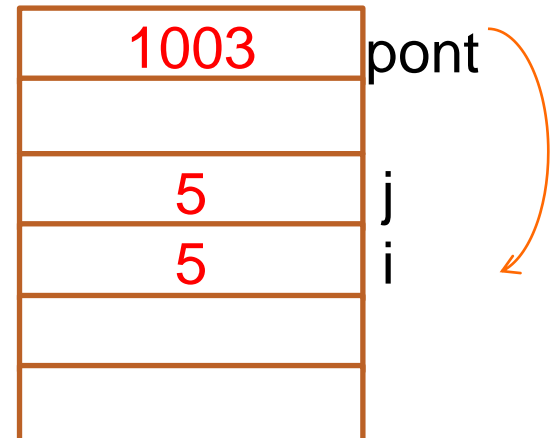
Devolve o conteúdo da variável cujo endereço é o operando imediatamente à sua direita, exemplo:

```
i = 5;  
pont = &i;  
j = *pont;
```

Endereço na
memória

1000
1001
1002
1003
1004
1005

Variável na
memória



Ponteiro

Variável pont armazena o endereço de um valor inteiro, que é representado pela variável i.

Assim a variável pont "aponta" para i.

Repare que o tipo da variável pont não é mais inteiro, e sim um endereço, um apontador para uma variável inteira.

Segue abaixo a declaração da variável pont:

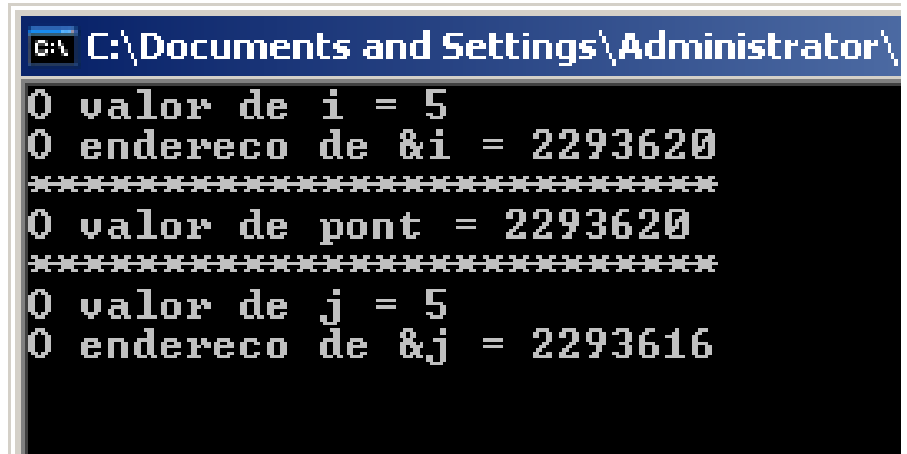
```
int *pont;
```

Essa declaração informa ao compilador que a variável pont irá armazenar um endereço de memória onde está armazenado um valor do tipo inteiro.

Digite o programa a seguir:

Ponteiro – Exemplo2

```
#include <stdio.h>
#include <conio.h>
main()
{
    int i,j,*pont;
    i = 5;
    pont = &i;
    j = *pont;
    printf("O valor de i = %d\n", i);
    printf("O endereco de &i = %d\n", &i);
    printf("*****\n");
    printf("O valor de pont = %d\n", pont);
    printf("*****\n");
    printf("O valor de j = %d\n", j);
    printf("O endereco de &j = %d\n", &j);
    getch();
}
```



```
C:\Documents and Settings\Administrator\
O valor de i = 5
O endereco de &i = 2293620
*****
O valor de pont = 2293620
*****
O valor de j = 5
O endereco de &j = 2293616
```

Exemplo3 – Atribuição de ponteiro de forma incorreta

```
#include <stdio.h>
#include <string.h>

main()
{
    float x=5, y;

    int *p;

    // ponteiro de inteiro (2 bytes)
    // apontando para float (8 bytes)
    p=&x;

    //não funciona como esperado
    y=*p;
    printf("%f\n", x);
    printf("%f", y);
}
```

Atribuição de Ponteiros

Pode ser usado no lado direito de um comando de atribuição:

```
#include <stdio.h>
#include <string.h>

main()
{
    int x=5;
    int *p1, *p2;

    p1 = &x;
    p2 = p1;
    //p1 e p2 armazenam o endereço de x

    printf("p1 = %p p2 = %p ", p1,p2);

}
```

Operações Aritmética com Ponteiros

São apenas duas possíveis, adição e subtração'.

Se considerar uma variável ponteiro de inteiro (2 bytes) **p1** com valor 5000.

Após: **p1++;**

p1 contém 5002 e não 5001. Pois o incremento é para o próximo elemento do mesmo tipo de base.

Também é possível apontar, por exemplo, para o décimo elemento do tipo **p1** adiante com o comando:

p1 = p1 + 10;

Indireção Múltipla (Ponteiros para ponteiros)



```
#include <stdio.h>
main()
{
    int x, *p, **q;
    x = 10;
    p = &x;
    q = &p;
    printf("  x = %d\n", x); // valor de x
    printf(" *p = %d\n", *p); // valor apontado por p
    printf("  p = %p\n", p); // valor de p (endereço x)
    printf("**q = %d\n", **q); // valor apontado por q
    // valor de p apontado por q (endereço x)
    printf(" *q = %p\n", *q);
    printf("  q = %p", q); // valor de q (endereço p)
}
```

Exercícios ponteiros

1) Quais serão os valores das variáveis x, y e p ao final do trecho de código:

```
int x, y=0, *p;  
p = &y;  
x = *p;  
x = 4;  
(*p)++;  
--x;  
(*p) += x;
```

2) O trecho de código a seguir possui um erro. Faça a correção.

```
void main() {  
int x, *p;  
x = 100;  
p = x;  
printf("Valor de p: %d.\n", *p);}
```

Exercícios ponteiros

- 3) Escreva um programa que contenha duas variáveis inteiras. Solicite os valores dessas variáveis para o usuário. Em seguida, compare seus endereços e exiba o conteúdo do maior endereço.
- 4) Escreva um programa que contenha um vetor de *float* com 10 elementos. Imprima o endereço de cada posição do vetor.
- 5) Escreva um programa que contenha uma matriz de *float* 3 x 3. Imprima o endereço de cada posição da matriz.

Funções

Funções

São blocos de instruções que realizam tarefas específicas.

Seu código é carregado uma vez e pode ser executado quantas vezes for necessário.

Com isso, os programas tendem a ficar menores e mais organizados, uma vez que o(s) problema(s) podem ser subdivididos em pequenas tarefas.

Funções

Quando se utiliza funções é possível realizar desvios na execução natural dos programas, pois geralmente são executados linearmente.

Os desvios são realizados quando uma função é chamada pelo programa principal.

Observe o exemplo a seguir.

Funções

```
1  ALGORITMO
2  DECLARE sal NUMÉRICO
3      LEIA sal
4      aum ← calculo (sal)
5      novo_sal ← sal + aum
6      ESCREVA "Novo salário é = ", novo_sal
7  FIM_ALGORITMO

8  SUB-ROTINA calculo (sal NUMÉRICO)
9      DECLARE perc, valor
10     LEIA perc
11     valor ← sal * perc / 100
12     RETORNE valor
13 FIM_SUB-ROTINA calculo
```

Funções em C

A linguagem C possibilita a modularização por meio das funções.

Programa escrito na linguagem C tem, no mínimo, uma função chamada **main()**, por onde começa a execução.

Existem também muitas outras funções predefinidas na linguagem C, por exemplo: `getch()`, `scanf()`, `printf()`.

Essas funções são adicionadas pela diretiva **#include**.

O usuário também pode criar quantas funções quiser, dependendo do problema que está sendo resolvido pelo programa.

Passagem de parâmetros e tipos de retorno

Funções podem receber vários valores, os parâmetros, e pode devolver um valor, o retorno.

Quando se especifica uma função deve-se deixar claro qual será o tipo de retorno e quais os parâmetros necessários para a execução da função.

Exemplo 1:

```
int soma(int a, int b)
{
    int s;
    s = a + b;
    return s;
}
```

parâmetros recebidos
(dois valores inteiros)




Variável “s” é privada da função soma



Resultado armazenado em s



Retorno ao programa
principal com o conteúdo
armazenado em s



Passagem de parâmetros e tipos de retorno

Exemplo 2:

```
float divisao(int dividendo, int divisor)
{
    float q;
    q = dividendo / divisor;
    return q;
}
```

← parâmetros recebidos
(dois valores inteiros)

← Variável “q” float é privada da
função divisao

← Resultado armazenado em q,
note que são tipos diferentes

← Retorno ao programa
principal com o conteúdo
armazenado em q

Exemplo sem retorno

Exemplo 3:

parâmetros recebidos
(dois valores inteiros)

```
void multiplicacao(float multiplicando, float multiplicador)
{
    float produto;
    produto = multiplicando * multiplicador;
    printf("O produto e = %d", produto);
    getch();
}
```

Não foi preciso retornar a variável pois utilizou seu conteúdo dentro da função.

Funções

Passagem de parâmetros por valor:

Para a execução da função, serão geradas cópias dos valores de cada um dos parâmetros.

Como exemplo, observe o programa a seguir:

Funções

```
1      #include <stdio.h>
2      #include <conio.h>
3      int soma_dobro(int a, int b);
4      main()    {
5      int x, y, res;
6      printf("\nDigite o primeiro numero: ");
7      scanf ("%d", &x);
8      printf("\nDigite o segundo  numero: ");
9      scanf ("%d", &y);
10     res = soma_dobro(x,y);
11     printf("\nSoma do dobro dos numeros %d e %d e = %d",x,y,res);
12     getch();   }

13     int soma_dobro(int a, int b)
14     {   int soma;
15         a = 2*a;
16         b = 2*b;
17         soma = a + b;
18         return soma;   }
```

Funções

Programa mostrado é um exemplo de uma passagem de parâmetros por valor.

Conteúdo das variáveis **x** e **y**, por meio da execução das linhas 7 e 9 do programa, por exemplo, foram 5 e 3.

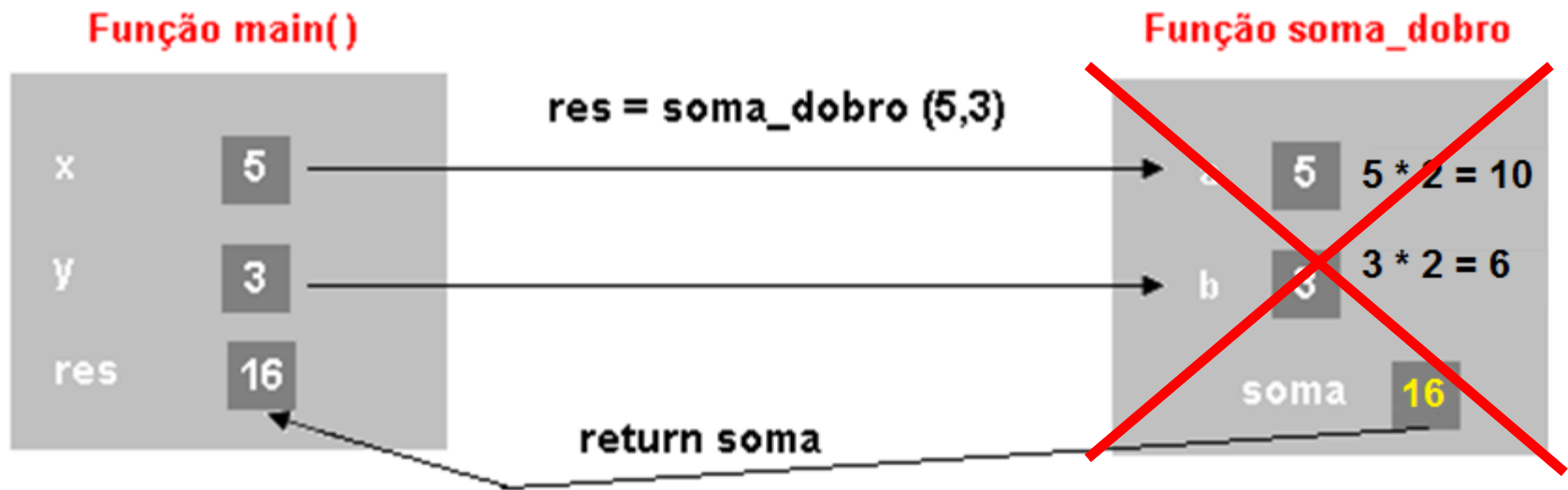
Na linha 10, os valores são copiados para **a** e **b** (pertencentes à função **soma_dobro**).

Depois, **a** e **b** são multiplicados por 2 e somados.

Linha 18, resultado (16) é devolvido à função **main** para a variável **res**.

No momento em que a função **soma_dobro** chega ao fim, as variáveis **a**, **b** e **soma** são destruídas e, portanto, as alterações realizadas pelas multiplicações por 2 são perdidas.

Funções



Após a linha 18

Funções

Passagem de parâmetros por referência:

Parâmetros passados para uma função correspondem a endereços de memória ocupados por variáveis.

Toda vez que for necessário acessar um determinado valor, isso será feito por meio de referência ao seu endereço.

Como exemplo, observe o programa a seguir:

Funções

```
1      #include <stdio.h>
2      #include <conio.h>
3      int soma_dobro(int *a, int *b);
4      main() {
5          int x, y, res;
6          printf("\nDigite o primeiro numero: ");
7          scanf ("%d", &x);
8          printf("\nDigite o segundo numero: ");
9          scanf ("%d", &y);
10         res = soma_dobro(&x,&y);
11         printf("\nSoma do dobro dos numeros %d e %d e = %d",x,y,res);
12         getch(); }

13     int soma_dobro(int *a, int *b)
14     {   int soma;
15         *a = 2*(*a);
16         *b = 2*(*b);
17         soma = *a + *b;
18         return soma; }
```

Funções

Linhas 7 e 9 são lidos os valores para as variáveis **x** e **y** (por exemplo 5 e 3).

Linha 10 chama a função **soma_dobro**, são passados como parâmetros para a função os endereços de memória ocupados pelas variáveis **x** e **y** (isso é feito pelo operador **&** - que obtém o endereço de memória de uma variável), por exemplo, 800 e 300.

a e **b** (da função) são ponteiros respectivamente, de **x** e **y**.

Linhas 15 e 16, os valores 5 e 3 são multiplicados por 2, no endereço 800 passamos a ter o valor 10 e no endereço 300 passamos a ter o valor 6.

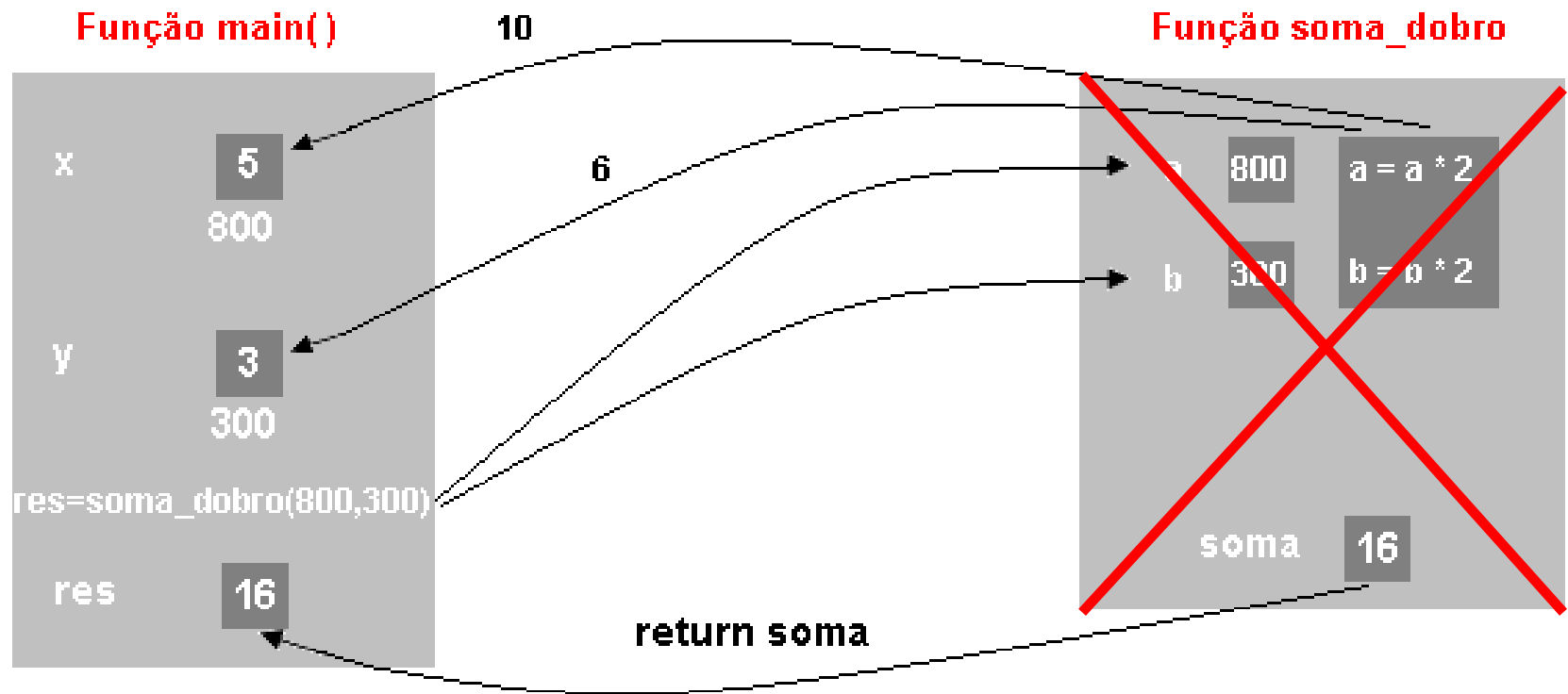
Funções

Linha 17 é realizada a soma dos valores que estão nos endereços especificados por **a** e **b** (que já foram multiplicados por 2).

Linha 18 o resultado da soma é devolvido à função **main**, atribuindo para a variável **res** e encerrando a função **soma_dobro**.

As alterações decorrentes das multiplicações feitas são mantidas, pois não geraram duplicatas de valores, mas sim, a todo momento fizeram referência a endereços de memória que estavam fora da área destinada à função.

Funções



Após a linha 18

Exercícios

- a) Faça um programa utilizando funções que leia um vetor de 5 números inteiros, depois imprime o vetor normal e o vetor invertido. Utilize funções para ler, imprimir e imprimir invertido os números do vetor.
- b) Faça um programa usando funções para calcular a média aritmética entre duas notas de um aluno e para mostrar a situação desse aluno, que pode ser aprovado ou reprovado.
- c) Faça um programa utilizando funções (com passagens de parâmetros por referência) que simula uma calculadora, o usuário digita 2 valores e depois escolhe a operação matemática que deseja fazer (soma, subtração, multiplicação, divisão).
- d) Complemente o exercício anterior com a criação de uma função com menu de opções para o usuário efetuar as operações matemáticas. O usuário digita 2 valores e de acordo com o menu, escolhe a operação desejada.