

Linguagem de Programação Imperativa

Introdução à Programação em Linguagem C

Carlos Mello
Hermes Camelo



Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

Iniciação à Ciência da Computação

Introdução

■ Bibliografia

- *Treinamento em Linguagem C*, Victorine Viviane Mizrahi, Vols 1 e 2

■ Ferramenta de Apoio

- Borland C++

Estrutura de um Programa

- Um programa em C consiste de uma ou mais funções.
 - Um programa em C pode conter dezenas, centenas, ou mais, funções com nomes únicos.
 - » Não confundir o termo função com funções matemáticas!!
 - Funções em C podem executar cálculos matemáticos;
 - Funções em C são conjuntos de instruções com um nome e que desempenham uma ou mais ações;
 - Funções em C não necessariamente devem retornar um valor.

Estrutura de um Programa

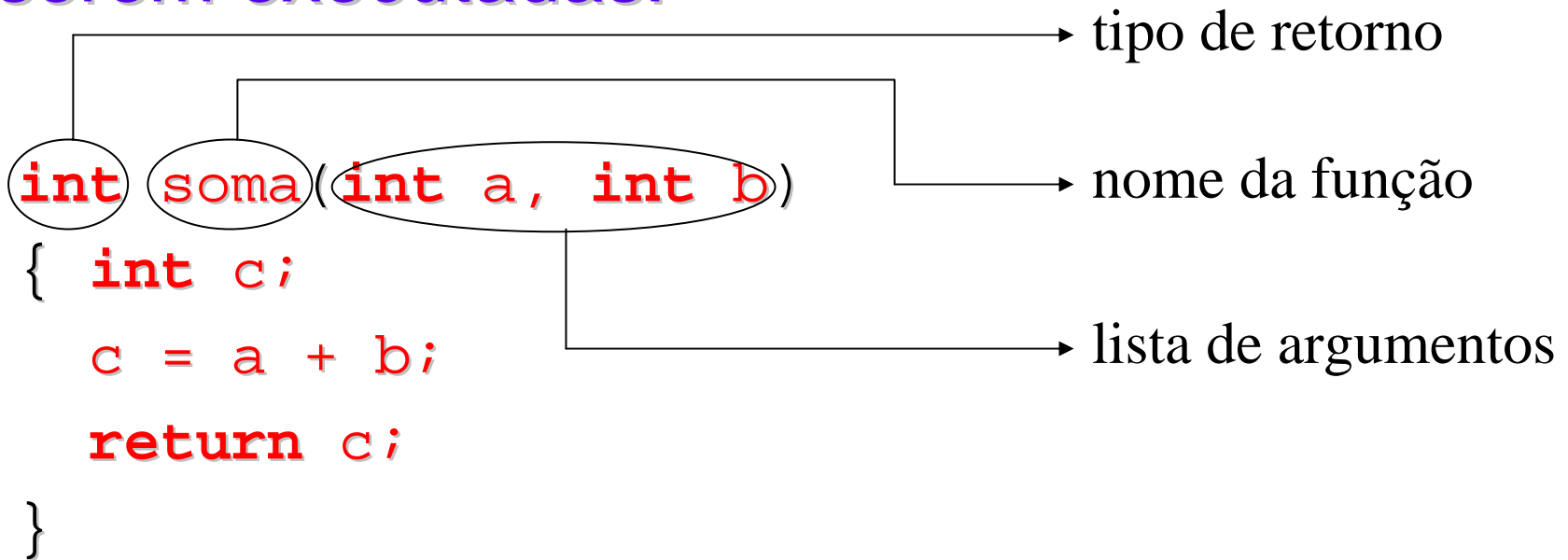
- Uma função em C sempre apresenta uma assinatura e um corpo com as sentenças a serem executadas:

```
int soma(int a, int b)
```

```
{ int c;  
  c = a + b;  
  return c;  
}
```

Estrutura de um Programa

- Uma função em C sempre apresenta uma assinatura e um corpo com as sentenças a serem executadas:



Estrutura de um Programa

- O ponto de partida para a execução do programa é controlado pela função `main()`:

```
main()  
{  
}
```

» Este é o menor programa em C!

Executando um Programa

- Escrever o programa em um arquivo texto (código fonte)
 - Salvar o arquivo com a extensão .c
- Compilar o programa fonte para gerar o código executável (*.exe)
 - » Um programa em C pode ser composto de vários códigos fontes (vários arquivos .c)
 - É comum a geração de um código objeto (*.obj) para cada código fonte e a posterior geração do código executável (linkedição).

Exemplos de Programa

```
main( )  
{ printf( "Bom Dia!" ); }
```

```
main( )  
{ int q;  
  q = 3 * 3;  
  printf( "O quadrado de 3 eh = %d", q );  
}
```


Uma Questão de Estilo

- Ao escrever seu programa, você pode colocar espaços, caracteres de tabulação e pular linhas à vontade, pois o compilador ignora estes caracteres.
 - Em C não há um estilo obrigatório.

Sentenças em C

- O corpo de uma função é composto de sentenças:
 - uma sentença pode ser uma declaração, uma instrução ou chamada a uma função:

```
int i, j;  
double x = 0, y = 0;  
j = 12; i = j + 27;  
x = soma(a,b);  
printf("Olá!!");  
if(i == x)  
    y = j / i;
```

Constantes e Variáveis

- Uma constante tem valor fixo e inalterável.
- Uma variável é um espaço vago, reservado e rotulado para armazenar dados.
 - Possui um nome que a identifica univocamente.
 - Possui um valor que corresponde à informação atribuída.
 - » O valor de uma variável pode mudar muitas vezes durante a execução de um programa, por meio de atribuições de valor.

Exemplo

```
main( )  
{ const double PI = 3.1415;  
  double raio, area;  
  raio = 4.0;  
  area = PI * raio * raio;  
  printf("A área de um círculo de ");  
  printf("raio %f = %f\n",raio,area);  
}
```

Tipos de Dados

■ Numéricos Inteiros

Tipo	Tamanho	Valores
char	8 bits	-128 a +127
short	16 bits	-32.768 a +32.767
int	32 bits	-2.147.483.648 a + 2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807

» +, -, *, /, ^, %

Tipos de Dados

■ Numéricos de Ponto Flutuante

Tipo	Tamanho	Valores
float	32 bits	$\pm 3.40282347\text{E}+38$ $\pm 1.40239846\text{E}-45$
double	64 bits	$\pm 1.79769313486231570\text{E}+308$ $\pm 4.94065645841246544\text{E}-324$

» 3.14159, 2.0, ...

» +, -, *, /, ...

Tipos de Dados

- **Números com e sem sinal**
 - C permite que o programador defina se uma variável de tipo numérico deva ou não reservar o bit de sinal (números negativos)
 - Notação:
 - » signed <tipo>
 - » unsigned <tipo>
 - Se nenhum modificador for indicado, o compilador C reservará o bit de sinal

Tipos de Dados Especiais

■ Booleano

- Qualquer variável inteira
 - » $0 \rightarrow$ Falso,
 - » $\neq 0 \rightarrow$ Verdadeiro
- são usados principalmente como o resultado de operadores relacionais
 - » $==, !=, >, <, >=, <=$
 - » `If (exp_bool) { exp } else { exp }`

Tipos de Dados Especiais

■ Caractere

- Representado pelo tipo `char`

- » 'a', 'b', '1', ' ', etc.

- internamente representa um código da tabela ASCII

Tipos de Dados Especiais

■ Sem Tipo

- Representado pela palavra `void`
- Não possui valor
- Mais utilizado em uma função para indicar que não possui valor de retorno (procedimento)

Declarando Variáveis

■ ATENÇÃO!

- Todo programa em C deve ter a função main()
 - » ela controla o fluxo de execução de todo o programa
 - um programa inicia com a primeira instrução de main() logo após a chave de abertura '{' e termina quando encontra a chave de fechamento '}'
- Um programa em C deve declarar todas as suas variáveis antes de usá-las.
 - » As declarações de variáveis devem vir antes das demais instruções
- Toda instrução em C é terminada por um ponto-e-vírgula.

Exercício

- Escreva um programa C que
 - declare 3 variáveis caractere e atribua a elas as letras a, b e c.
 - Declare também 3 variáveis inteiras e atribua os valores 1, 2 e 3 a elas.
 - Declare uma variável de ponto flutuante para guardar a média aritmética das 3 variáveis inteiras declaradas anteriormente.

Identificadores

- Identificadores são símbolos singulares utilizados nos programas.
 - O nome da função `main()` é um identificador.
 - O nome de uma variável ou de uma constante é um identificador.
 - Identificadores bem escolhidos são sua ferramenta principal para escrever programas que façam sentido.

Identificadores

- A criação e uso de identificadores fica a cargo do programador.
 - Identificadores podem conter apenas letras minúsculas ou maiúsculas, algarismos e traço de sublinha. Além disso, identificadores não devem começar com algarismos.

Identificadores

- Quais dos seguintes nomes são válidos para identificadores em C?

3ab

_sim

n_a_o

não

char

AuToCaD

*abc

guarda-valor

\ontem

a1

cem**anos

dez^anos

A1

xFuncao

C&A

Identificadores

- Quais dos seguintes nomes são válidos para identificadores em C?

3ab

_sim

n_a_o

não

char

AuToCaD

*abc

guarda-valor

\ontem

a1

cem**anos

dez^anos

A1

xFuncao

C&A

Identificadores

- C é sensível ao tipo de letra utilizado (distingue maiúsculas de minúsculas) em todas as letras do identificador.
 - Isso quer dizer que `minhaFuncao`, `minhafuncao` e `MinhaFuncao` são identificadores *diferentes*.
 - » Por essa razão, vale a pena adotar um estilo de digitação e se manter coerente a ele.

Palavras Reservadas

■ Algumas palavras especiais da linguagem C

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

Definindo Funções

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

void main()
{
    printf("A soma de 3 com 4 = %d\n", soma(3,4) );
}
```

programa.c

Definindo Funções

```
float soma(float a, float b);

void main()
{ printf("A soma de 3 com 4 = %f\n", soma(3,4) );
}

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

programa.c

A Função printf()

- Utilizada para enviar sequências de caracteres para a saída padrão de um programa em C
- Sintaxe:
 - `printf("string de controle", lista de argumentos);`

A Função printf()

■ Exemplos:

- `printf("Uma linha");`
- `printf("Uma linha\nDuas linhas");`
- `printf("Os números são: %d e %d\n", 7, 8);`
- `printf("%s está a %d Km de Recife",
"Caruaru", 120);`
- `printf("%d%% de %d = %f\n", p, x,
x*(p/100.0));`

A Função printf()

■ Códigos especiais:

<code>\n</code>	avanço de linha
<code>\t</code>	tabulação (tab)
<code>\b</code>	retrocesso (backspace)
<code>\"</code>	aspas duplas
<code>\\</code>	barra

A Função printf()

■ Códigos de impressão formatada:

<code>%c</code>	caractere
<code>%d</code>	inteiro
<code>%e</code>	notação científica
<code>%f</code>	ponto-flutuante
<code>%o</code>	octal
<code>%x</code>	hexadecimal
<code>%u</code>	inteiro sem sinal
<code>%s</code>	string
<code>%%</code>	o caractere <code>'%'</code>

A Função printf()

- O tamanho de campos de impressão é indicado logo após o '%' e antes do tipo do campo:

- `printf("Os alunos são %4d!\n", 44);`
- `printf("Os alunos são %04d!\n", 44);`
- `printf("Os alunos são %-4d!\n", 44);`
- `printf("R$ %.2f!\n", 1234.5632);`
- `printf("R$ %10.2f!\n", 1234.5632);`
- `printf("R$ %-10.2f!\n", 1234.5632);`

Exercício

- Escreva um programa que imprima na tela:

um

dois

três

quatro

Iniciação à Ciência da Computação

Programação em Linguagem C

Entrada de Dados e Operadores

Carlos Mello

Hermes Camelo

Ricardo Massa



Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

A Função scanf()

- semelhante à função printf(), exceto que é utilizada para leitura de dados
- Sintaxe:
scanf("string de controle", endereço dos argumentos);
- Exemplo:

```
int anos;
```

```
printf("Digite sua idade em anos: ");
```

```
scanf("%d", &anos);
```

→ endereço da variável

Endereços de Variáveis

- Um endereço de memória é visto como um número inteiro sem sinal
- O código para formatação de um endereço é %u

- Exemplo:

```
int num = 2;
```

```
printf("Valor=%d, endereço=%u", num, &num);
```

Exercício

- Escreva um programa que solicite a idade de uma pessoa em anos e converta a idade para dias.

As funções `getche()` e `getch()`

- Em algumas situações, a função `scanf()` não se adapta perfeitamente pois é preciso pressionar <enter> depois da entrada
- As funções `getche()` e `getch()` efetuam a leitura de um caractere e continuam a execução do programa
 - `getche()` apresenta o caractere lido na tela, enquanto que `getch()` não apresenta

Exercício

- Escreva um programa que peça para o usuário digitar um caractere na tela, imprima o caractere digitado na mesma linha e, por fim, imprima em linhas diferentes:
 - seu valor na tabela ASCII
 - seu antecessor e o valor dele na tabela ASCII
 - seu sucessor e o valor dele na tabela ASCII

Use as funções `getche()` e `getch()`.

As funções `getchar()` e `putchar()`

- A função `getchar()` lê o primeiro caractere de um string e termina quando a tecla `<enter>` for pressionada
- A função `putchar()` imprime um caractere na tela.

Exercício

- Escreva um programa que peça ao usuário para escrever um string na tela e que escreva o primeiro caractere do string na linha seguinte.

Operadores Aritméticos

■ Operadores Binários

=	atribuição
+	adição
-	subtração
/	divisão
%	resto da divisão (módulo)

■ Operador Unário

-	menos unário
---	--------------

Operador de Atribuição

- O operador de atribuição não tem equivalente na matemática
- Exemplo: `num = 2000;`
 - atribui o valor 2000 à variável `num`
 - `2000 = num;` não faz sentido em C!
- C aceita várias atribuições em uma mesma instrução
 - `a = b = c = 10;`

Exercício

- Escreva um programa que solicite ao usuário uma temperatura em graus Fahrenheit e imprima o equivalente em graus Celsius

$$\text{Celsius} = (\text{Fahrenheit} - 32) * 5 / 9$$

Incremento e Decremento

- ++ soma 1 ao seu operando
- -- subtrai 1 do seu operando

- Exemplo:

```
int num1, num2;  
num1 = 5;  
num2 = num1++;  
printf("num1=%d, num2=%d", num1, num2);
```

- O resultado será:

```
num1=6, num2=5
```

Incremento e Decremento

- Se o operador for pré-fixado, o resultado será diferente.

- Exemplo:

```
int num1, num2;
```

```
num1 = 5;
```

```
num2 = ++num1;
```

```
printf("num1=%d, num2=%d", num1, num2);
```

- O resultado será

```
num1=6, num2=6
```

Operadores Aritméticos de Atribuição

<code>num += 2</code>	equivale a	<code>num = num + 2</code>
<code>num -= 2</code>	equivale a	<code>num = num - 2</code>
<code>num *= 2</code>	equivale a	<code>num = num * 2</code>
<code>num /= 2</code>	equivale a	<code>num = num / 2</code>
<code>num %= 2</code>	equivale a	<code>num = num % 2</code>

Operadores Aritméticos de Atribuição

<code>x *= y + 1</code>	equivale a	<code>x = x * (y+1)</code>
<code>t /= 2.5</code>	equivale a	<code>t = t/2.5</code>
<code>p %= 5</code>	equivale a	<code>p = p%5</code>

Qual será o valor de x, y e z?

```
int x=1, y=2, z=3;
```

```
x += y += z += 7;
```

Escreva um programa para testar sua resposta.

Precedência

- Alguns operadores tem uma prioridade maior de execução que outros.
- Qual será o valor de cada variável abaixo?
(assuma que todas são do tipo int)

`x = (2+1)*6;`

`y = (5+1)/2*3;`

`i = j = (2+3)/4;`

`a = 3+2*(b=7/2);`

`c = 5+10%4/2;`

Cuidados com C !!

Type Casting - Coerção

- Cuidado com o uso de variáveis de tipos diferentes em C:
 - Teste o código abaixo e veja o resultado....

```
main( )  
{  
    int x,y;  
    float z;  
  
    x=2;  
    y=3;  
    z=(x+y)/2;  
    printf ("z = %f\n", z);  
}
```

Cuidados com C !!

Type Casting - Coerção

- Modifique o código anterior para...

```
main( )
{
    int x,y;
    float z;

    x=2;
    y=3;
    z=((float) (x+y))/2;
    printf ("z = %f\n", z);
}
```

Exercício

- O programa a seguir tem vários erros em tempo de compilação. Verifique e corrija-os.

```
Main( )  
{  
    int a=1; b=2, c=3;  
    printf("Os números são: %d %d %d\n,a,b,c,d);  
}
```

Exercício (resposta)

- O programa a seguir tem vários erros em tempo de compilação. Verifique e corrija-os.

```
main( )
{
    int a=1, b=2, c=3;
    printf("Os números são: %d %d %d\n",a,b,c);
}
```

Exercício

- Qual a execução do programa a seguir?

```
void main()  
{ int n;  
  printf("Digite um número inteiro: ");  
  scanf("%d",&n);  
  printf("Os números são: %d %d %d\n",  
                                                n, n+1, n++);  
}
```

```
> programa  
Digite um número inteiro: 5  
Os números são: 6 7 5
```

Exercício

■ Teste agora

```
void main()  
{ int n;  
  printf("Digite um número inteiro: ");  
  scanf("%d",&n);  
  printf("Os números são: %d %d %d\n",  
                                                n, n+1, ++n);  
}
```

```
> programa  
Digite um número inteiro: 5  
Os números são: 6 7 6
```


Iniciação à Ciência da Computação

Programação em Linguagem C

Instruções de Controle: Tomando Decisões

Carlos Mello

Hermes Camelo

Ricardo Massa

Engenharia da Computação

Escola Politécnica de Pernambuco - UPE



Instruções de Controle

- Toda linguagem de programação precisa oferecer pelo menos três formas básicas de controle:
 - executar uma sequência de instruções
 - realizar testes para decidir entre ações alternativas
 - repetir uma sequência de instruções

Tomando Decisões

- A realização de testes para decidir entre ações alternativas envolve o uso de operadores lógicos e relacionais, cujos resultados vão guiar o fluxo de execução de um programa

SE x == 0 ENTÃO

IMPRIMA "Divisão por zero!"

SENÃO

IMPRIMA (y / x)

Expressão booleana.
Valor de retorno:
Verdadeiro ou Falso

Operadores Relacionais

- Retornam um valor booleano

>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
==	igual
!=	diferente

Operadores Lógicos

- Operam sobre valores booleanos

&& E (0 && 1 == 0)

|| OU (0 || 1 == 1)

! NÃO (!0 == 1)

- Exemplos:

```
int a = 2, b = 5;
```

```
char v1 = (a > 0) && (b != a);
```

```
char v2 = !v1;
```

```
char v3 = !(a < 0); /* (a >= 0) */
```

Comandos de Decisão

- C oferece 4 estruturas de decisão:
 - if
 - if-else
 - switch
 - operador condicional

Tomando Decisões

- **if e if-else**

if(expressão_de_teste)

comando ou bloco 1

else

comando ou bloco 2

```
if(a > b)
```

```
    printf("a é maior que b");
```

```
else
```

```
{ c = b - a;
```

```
    printf("b é maior ou igual a a");
```

```
}
```

Exemplo

```
char ch = getche();
```

```
if(ch == 'p')
```

```
{ printf("\n Você digitou a tecla 'p'");  
  printf("\n Digite qualquer tecla ");  
  printf("para terminar...");  
  getch();  
}
```


Exemplo 2

```
char ch = getche();
```

```
if(ch == 'p')
```

```
    printf("\n Você digitou a tecla 'p'");
```

```
else
```

```
    printf("\n Você digitou a tecla '%c'",ch);
```

```
printf("\n Digite qualquer tecla ");
```

```
printf("para terminar...");
```

```
getch();
```

Exercício

- Desenvolva um programa para implementar uma calculadora com quatro operações utilizando comandos `if-else` para identificar qual das quatro operações deve ser realizada.

Exercício (Resposta)

```
float num1, num2;
char oper;
printf("Digite: número operador número\n");
scanf("%f %c %f", &num1, &oper, &num2);
if(oper == '+')
    printf("= %f\n", num1+num2);
else if(oper == '-')
    printf("= %f\n", num1-num2);
else if(oper == '*')
    printf("= %f\n", num1*num2);
else if(oper == '/')
    printf("= %f\n", num1/num2);
else
    printf("\nOperação não reconhecida!!!\n");
```

Tomando Decisões

- O comando `if-else` é útil para a escolha de uma entre duas alternativas
- Quando mais de duas alternativas são necessárias, pode ficar deselegante utilizar vários `if-else` encadeados
 - Para estes casos o comando `switch` é a melhor opção

Tomando Decisões

■ switch

```
switch(expressão_constante)
{
    case constante1:
        comando ou bloco 1
        break;

    ...

    default:
        comando ou bloco
}
```

Exemplo

```
int codigo;
printf("Digite o código da operação: ");
scanf("%d", &codigo);
switch(codigo)
{ case 1 : printf("Extrato de Conta Corrente");
          tira_extrato();
          break;
  case 2 : printf("Transferência");
          transfere_dinheiro();
          break;
  default: printf("Código inválido");
}
```

Exemplo 2

```
float total, a, b;
char operador;
printf("Digite: número operador número\n");
scanf("%f %c %f", &a, &operador, &b);
switch(operador)
{
  case '+': total = a + b; break;
  case '-': total = a - b; break;
  case '*':
  case 'x': total = a * b; break;
  case '/': total = a / b; break;
  default: printf("Operador desconhecido\n");
           total = 0.0;
}
printf("Total da operação = %f\n", total);
```

Tomando Decisões

■ Expressão condicional (? :)

*expressão_de_teste ? expressão1 :
expressão2;*

A *expressão1* será avaliada caso *expressão_de_teste* seja verdadeira. Caso contrário, *expressão2* será avaliada

```
int a, b, maior;  
a = 17 + 15;  
b = 3 * 7;  
maior = (a > b) ? a : b;
```


Exercício

- O que será impresso pelo programa a seguir?

```
void main()  
{ int num = -42;  
  printf("O valor é %d",  
        (num<0) ? 0 : num*num );  
}
```

Iniciação à Ciência da Computação

Programação em Linguagem C

Instruções de Controle: Laços

Carlos Mello

Hermes Camelo

Ricardo Massa



Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

Instruções de Repetição

- Laços são utilizados para repetir uma sequência de instruções.
- Exemplo:

ENQUANTO houver refrigerantes FAÇA
pergunte qual refrigerante o cliente deseja
receba o dinheiro
forneça o refrigerante
devolva o troco

Instruções de Repetição

- A linguagem C oferece 3 tipos de laços:
 - for
 - while
 - do-while
- » todos eles fazem a mesma coisa, ou seja, executa uma mesma sequência de instruções sempre que uma condição for satisfeita

Instruções de Repetição

- O laço `for` engloba 3 expressões
 - **inicialização**
 - » executada uma única vez no início do laço
 - **teste**
 - » condição que controla o laço; o laço será executado enquanto esta condição for verdadeira
 - **incremento**
 - » define como a variável de controle do laço será alterada

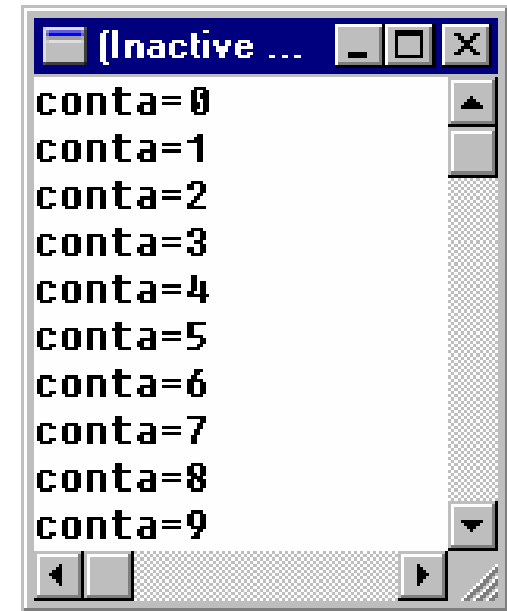
Instruções de Repetição

■ O laço *for*

for(inicialização; teste; incremento)
comando ou bloco

```
for(conta=0; conta<10; conta++)  
    printf("conta=%d\n", conta);
```

instrução a ser repetida



The screenshot shows a terminal window titled "(Inactive ...". It displays the output of the program, which is a list of numbers from 0 to 9, each preceded by the text "conta=" and followed by a newline character. The output is as follows:

```
conta=0  
conta=1  
conta=2  
conta=3  
conta=4  
conta=5  
conta=6  
conta=7  
conta=8  
conta=9
```

Exercício

- **Faça um programa que imprima em ordem decrescente todos os valores inteiros maiores que zero a partir de um número fornecido pelo usuário**

Exercício (Resposta)

```
void main()  
{ int c, n;  
  printf("Forneça um inteiro positivo: ");  
  scanf("%d",&n);  
  if(n >= 0)  
  { for(c = n; c > 0; c--)  
    printf("%d ",c);  
  }  
  else  
    printf("Valor negativo.\n");  
}
```


Exemplo

inicialização ponto-e-vírgula teste incremento não coloque ponto-e-vírgula aqui

```
int conta, total;  
for(conta = 0, total = 0; conta < 10; conta++)  
{  
    total += conta;  
    printf("conta = %d, total = %d\n", conta, total);  
}
```

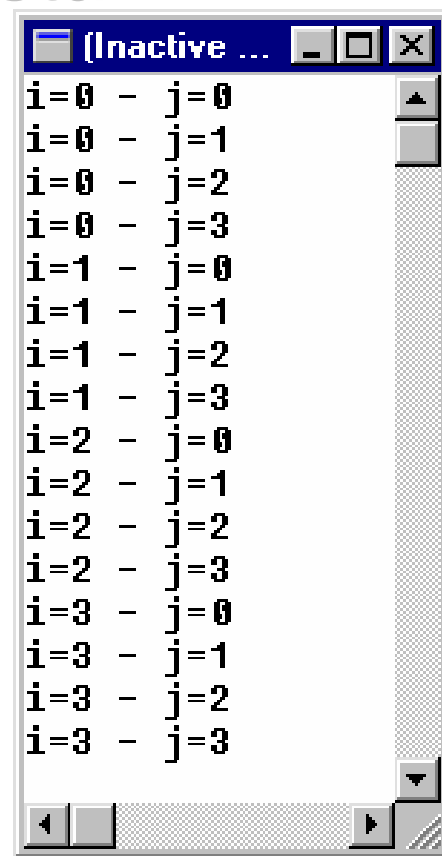
corpo do laço

conta = 0, total = 0
conta = 1, total = 1
conta = 2, total = 3
conta = 3, total = 6
conta = 4, total = 10
...

Laços Aninhados

- Quando um laço está dentro do escopo de outro, diz-se que o laço interior está aninhado

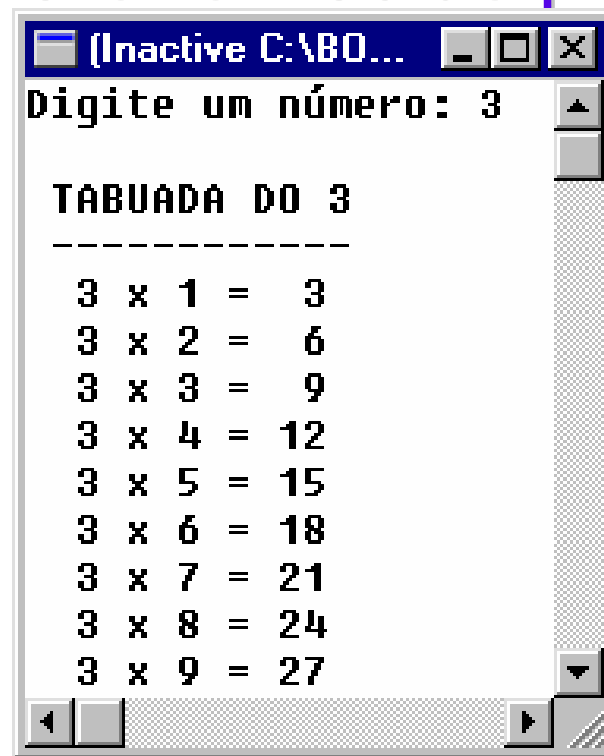
```
for(i = 1; i < 4; i++)  
    for(j = 1; j < 4; j++)  
        printf("i=%d - j=%d\n", i, j);
```



```
(Inactive ...  
i=0 - j=0  
i=0 - j=1  
i=0 - j=2  
i=0 - j=3  
i=1 - j=0  
i=1 - j=1  
i=1 - j=2  
i=1 - j=3  
i=2 - j=0  
i=2 - j=1  
i=2 - j=2  
i=2 - j=3  
i=3 - j=0  
i=3 - j=1  
i=3 - j=2  
i=3 - j=3
```

Exercício

- Faça um programa para imprimir a tabuada de um número fornecido pelo usuário



```
(Inactive C:\BO...
Digite um número: 3

TABUADA DO 3
-----
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
```

Instruções de Repetição

■ O laço *while*

`while(expressão_de_teste)`
comando ou bloco

```
int c, total=0;
c = 0;
while(c < 10)
{ printf("c é igual a %d\n",c);
  c+=17;
  total++;
}
printf("O laço foi executado %d vezes.\n",total);
```

while x for

```
for(conta=0; conta<10; conta++)  
    printf("conta=%d\n", conta);
```

```
conta=0;  
while(conta<10)  
{  
    printf("conta=%d\n", conta);  
    conta++;  
}
```

Equivalentes

Exercício

- Implemente a função fatorial de forma iterativa. Escreva duas versões para ela: uma usando o laço for e outra usando o laço while.
- Escreva a função `main()` que fique solicitando números ao usuário e imprimindo o fatorial de cada um deles. O programa será finalizado quando o usuário fornecer um número negativo.

while x for

- O laço `for` é mais indicado quando o número de iterações for conhecido antecipadamente.
- O laço `while` é mais apropriado quando a iteração possa ser terminada inesperadamente, em consequência das operações do corpo do laço.

Exemplo

```
void main()  
{ int cont, letra, totPal;  
  printf("Digite uma frase:\n");  
  cont = 0;  totPal = 0;  
  while( (letra=getche()) != '.' )  
  { cont++;  
    if(letra==' ') totPal++;  
  }  
  if(cont>0) totPal++;  
  printf("\n");  
  printf("Total de caracteres: %d\n",cont);  
  printf("Total de palavras: %d\n",totPal);  
}
```


Instruções de Repetição

■ O laço *do/while*

```
do
{ comandos
}while(expressão de teste);
```

ponto-e-vírgula aqui

É semelhante ao while, porém testando a condição após a execução do laço. Dessa forma, o corpo do laço sempre é executado pelo menos uma vez.

Exemplo

```
int num;
do
{ printf( "Digite um número para calcular " );
  printf( "seu fatorial. Digite um número " );
  printf( "negativo para finalizar: " );
  scanf( "%d", num );
  printf( "O fatorial de %d é %d\n",
          num, fat( num ) );
}while( num >= 0 );
```

do-while

- Estimativas indicam que o laço `do-while` é necessário em apenas 5% dos laços
 - ler a expressão de teste antes de percorrer o laço ajuda o leitor a interpretar o sentido do bloco de instruções.

break

- A execução do comando `break` causa a saída imediata do laço
 - O comando `break` pode ser usado no corpo de qualquer laço
 - Se o `break` estiver em laços aninhados, afetará somente o laço mais interno

continue

- O comando `continue` força a próxima iteração do laço e pula o código que estiver abaixo.
 - deve ser evitado pois pode causar dificuldade de leitura e confusão ao se manter o programa

Exemplo

```
int num;
while (1)
{ printf( "\n Digite um número maior que 0:" );
  scanf( "%d", &num );
  if( num < 0 )
  { printf( "número errado\n" );
    continue;
  }
  printf( "Número correto" );
  if( num > 100 )
    break;
}
```

Exercício

- Crie um jogo em que o jogador tenta adivinhar uma letra entre **a** e **z** gerada aleatoriamente pelo programa.
 - Quando o jogador digitar uma letra errada, informe que ele errou e peça para ele tentar novamente.

Exercício (cont)

- Quando o jogador acertar a letra, informe o número de tentativas e pergunte se ele deseja continuar jogando.
 - » Caso ele responda que sim, gere uma nova letra e inicie uma nova partida. Caso contrário, finalize o jogo.
- Use o comando `ch=rand() % 26 + 'a'` para gerar a letra aleatoriamente.

Exercício (cont)

- Uma execução do programa seria:

```
Digite uma letra entre 'a' e 'z'  
w é incorreto. Tente novamente.  
k é incorreto. Tente novamente.  
a é incorreto. Tente novamente.  
i é correto.
```

```
Você acertou em 4 tentativas
```

```
Quer jogar novamente (s/n): n
```

Exercício (Resposta)

```
char c, ch;
int tentativas;
do
{
    ch = rand()%26 + 'a';
    tentativas = 1;
    printf("\nDigite uma letra de 'a' a 'z': ");
    while((c=getch( ))!=ch)
    {
        printf("\n%c é incorreto. Tente novamente.\n", c);
        tentativas++;
    }
    printf("\n %c é correto",c);
    printf("\n você acertou em %d tentativas",
                                                tentativas);

    printf("\n Deseja jogar novamente ? (s/n): ");
} while(getche( ) == 's');
```

Iniciação à Ciência da Computação

Programação em Linguagem C

Funções

Carlos Mello

Hermes Camelo

Ricardo Massa

Engenharia da Computação

Escola Politécnica de Pernambuco - UPE



Função

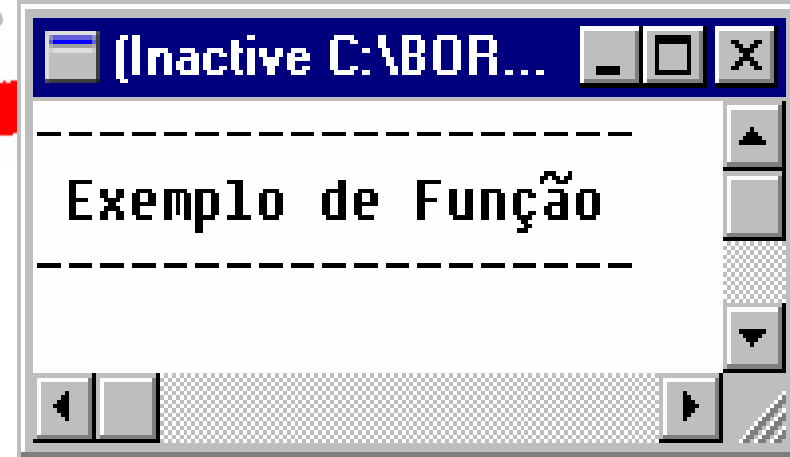
- Unidade autônoma do programa desenvolvida para executar alguma atividade
- Exemplos:
 - `getch()`
 - `printf()`
 - `scanf()`
 - `getchar()`
 - `rand()`

Criando Funções

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

```
void linha( )
{ int cont;
  for(cont=1;cont<=19;cont++) printf( "-" );
  printf( "\n" );
}

void main( )
{ linha( );
  printf( " Exemplo de Função \n" );
  linha( );
}
```



Criando Funções

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

```
#include <conio.h>
```

```
void linha( )
```

```
{ int cont;
```

```
  for(cont=1;cont<=19;cont++) printf( "- " );
```

```
  printf( "\n" );
```

```
}
```

```
void main( )
```

```
{ linha( );
```

```
  printf( " Exemplo de Função \n" );
```

```
  linha( );
```

```
}
```

Os parênteses indicam que é uma chamada a uma função

Escopo de Variáveis

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

```
int total;
```

→ global

```
void linha(char sep)
```

```
{ int cont;
  for(cont=1;cont<=19;cont++) printf("%c",sep);
  total = cont;
}
```

→ local

```
void main( )
```

```
{ total = 3;
  linha('=');
  printf("\nTotal = %d\n",total);
}
```

Escopo de Variáveis

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

```
int total;
```

→ global

```
void linha(char sep)
```

```
{ int cont;
```

→ local

```
  for(cont=1; cont<=19; cont++) printf("%c", sep);  
  total = cont;  
}
```

```
void main( )
```

```
{ total = 3;  
  linha('=');  
  printf("\nTotal = %d\n", total);  
}
```


Escopo de Variáveis

```
#include <stdio.h>
int total;
void linha(char sep)
{ int cont;
  for(cont=1;cont<=19;cont++) printf("%c",sep);
  total = cont;
}
void main( )
{ int cont = 5;
  total = 3;
  linha('=');
  printf("\nTotal = %d\n",total+cont);
}
```

variáveis diferentes!!



Exemplo de Funções

```
int fat(int n)
{ if(n<=0) return 1;
  else      return n*fat(n-1);
}
```

```
void tabelaASCII()
{ unsigned char c;
  printf("Código  Letra\n");
  printf("-----\n");
  for(c = 32; c < 255; c++)
    printf(" %4d      %c\n",c,c);
}
```

Exemplo

- Em um arquivo chamado `func.c`, crie três funções:
 - `int quad(int x)` que retorna o quadrado de um número
 - `int cubo(int x)` que retorna o cubo de um número
 - `char ehPrimo(int x)` que retorna 1 (*verdadeiro*) caso o número seja primo ou 0 (*falso*) caso contrário

Resposta

```
int quad(int x)
{ return x*x; }
```

```
int cubo(int x)
{ return x*x*x; }
```

```
char ehPrimo(int x)
{ int c, resp = 1;
  for(c=2; resp && c < x; c++)
    if( (x%c) == 0)
      resp = 0;
  return resp;
}
```

Exemplo

- Agora crie um arquivo chamado `ex01.c` que contenha a função `main()` que irá:
 - Solicitar um número n ao usuário e imprimir o quadrado de todos os números primos de 1 a n e o cubo de todos os números de 1 a n que não forem primos.
 - Use as funções que estão no arquivo `func.c`

Resposta

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

```
#include "func.c" ←
```

```
void main()
```

```
{ int n, c;
```

```
printf("Escreva um número: "); scanf("%d",&n);
```

```
for(c=1; c<=n; c++)
```

```
{ if(ehPrimo(c))
```

```
    printf("O quadrado de %d é %d\n",c,quad(c));
```

```
else
```

```
    printf("O cubo de %d é %d\n",c,cubo(c));
```

```
}
```

```
}
```

Exercício

- Escreva uma função recursiva em C para calcular o fatorial de um número
 - utilize o comando `if` para testar se o número solicitado é maior ou igual a 1
- Escreva um programa em C que use a função definida anteriormente para imprimir o fatorial de um número indicado pelo usuário

Iniciação à Ciência da Computação

Programação em Linguagem C

Vetores

Carlos Mello

Hermes Camelo

Ricardo Massa

Engenharia da Computação
Escola Politécnica de Pernambuco - UPE



Vetores

■ Observe:

```
int nota1, nota2, nota3;
printf("Digite a nota do aluno 1: ");
scanf("%d",&nota1);
printf("Digite a nota do aluno 2: ");
scanf("%d",&nota2);
printf("Digite a nota do aluno 3: ");
scanf("%d",&nota3);
printf("A média dos alunos é: %.2f",
      (nota1+nota2+nota3)/5.0);
```

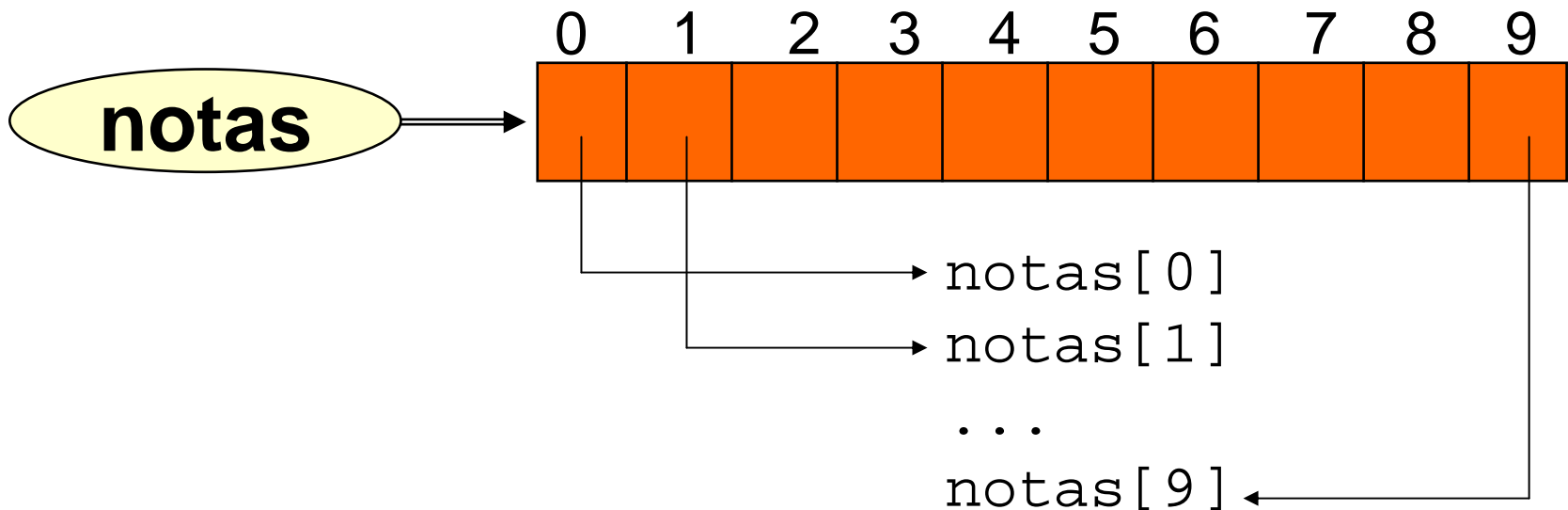
Vetores

- O que acontece se você desejar encontrar a média dos 2000 alunos de uma escola?
- Solução: usar um vetor com 2000 itens
 - Um vetor é um tipo de dado utilizado para representar um conjunto de valores homogêneos utilizando um único nome.
 - Cada valor é diferenciado através do índice do vetor
 - Em C, o primeiro elemento tem índice 0.

Vetores

```
int notas[10];
```

- Aloca 10 valores inteiros referenciados através do identificador **notas**.



Exemplo

```
int notas[2000];
int soma, i;
for(i = 0; i < 2000; i++)
{ printf("Digite a nota do aluno %d: ", i);
  scanf("%d", &notas[i]);
}
soma = 0;
for(i = 0; i < 2000; i++)
    soma = soma + notas[i];
printf("Média dos alunos = %.2f\n",
                                             soma/2000.0);
```

Cuidados

- C não avisa quando o limite de um vetor é excedido!
 - Se o programador transpuser o fim do vetor durante a operação de atribuição, os valores serão armazenados em outros dados ou mesmo no código do próprio programa.
- O programador tem a responsabilidade de verificar o limite do vetor

Inicializando Vetores

```
int tab[5] = {7, 0, -9, 15, 38};  
int fib[] = {1,1,2,3,5,8,13,21,33};
```

- Na ausência do tamanho do vetor, o compilador contará o total de itens na lista de inicialização
- Se o programador declarar um vetor sem inicializá-lo, ele deverá declarar sua dimensão
 - » Se há menos inicializadores que a dimensão especificada, os outros serão zero
 - » Mais inicializadores que o necessário implica em erro

Strings

- Podemos armazenar uma sequência de caracteres (string) em um vetor:

```
char nome[5];
```

- Como C não controla automaticamente o limite do vetor, sempre devemos sinalizar o final do string com o caractere especial `'\0'`

```
nome[0] = 'c';  nome[1] = 'a';
```

```
nome[2] = 's';  nome[3] = 'a';
```

```
nome[4] = '\0';
```

Strings constantes

```
printf( "Um string constante!\n" );  
printf( "%s fica muito longe", "Plutão" );
```

ERRADO:

```
char nome[10] = { "Corrida" };  
nome = "Viagem";
```

CORRETO:

```
char nome[10] = "Corrida";  
char nome[10] =  
    { 'C', 'o', 'r', 'r', 'i', 'd', 'a', '\0' };
```


Funções para Strings

■ Definidas em `string.h`:

- `strcpy(char *destino, char *fonte);`
- `strcat(char *destino, char *fonte);`
- `strlen(char *fonte);`
- `sprintf(char *destino, char *controle,...);`
- `gets(char *destino);`
- `puts(char *fonte);`

Exemplo

```
char nome[21];
int ano[2];
printf("Qual é seu nome? ");
gets(nome);
printf("%s, em que ano estamos? ", nome);
scanf("%d", &ano[0]);
printf("%s, em que ano você nasceu? ", nome);
scanf("%d", &ano[1]);
printf("%s, sua idade é %d anos.\n", nome,
                                             ano[0]-ano[1]);
```

Exemplo

```
char msg[81], nome[21], sobrenome[21];
int idade;
printf("Qual é seu nome? ");
gets(nome);
printf("Qual é seu sobrenome? ");
gets(sobrenome);
printf("Qual é sua idade? ");
scanf("%s", &idade);
strcpy(msg,nome);
strcat(msg, " ");
strcat(msg,sobrenome);
sprintf(msg,"%s tem %d anos de idade",msg,idade);
puts(msg);
```

Iniciação à Ciência da Computação

Programação em Linguagem C

Ponteiros

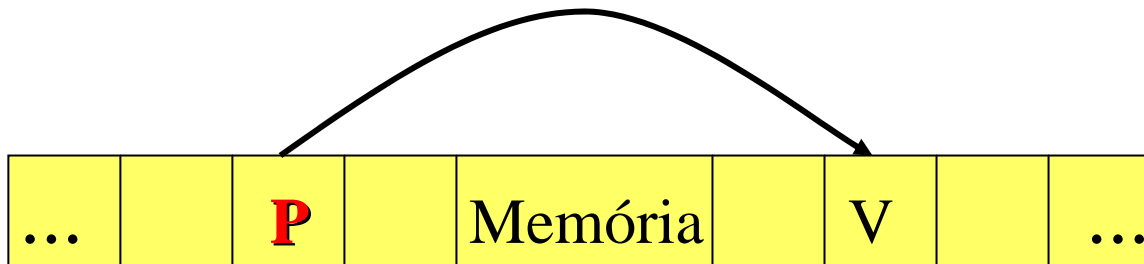
Carlos Mello
Ricardo Massa



Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

Ponteiros

- Um ponteiro é uma variável que contém o endereço de outra variável



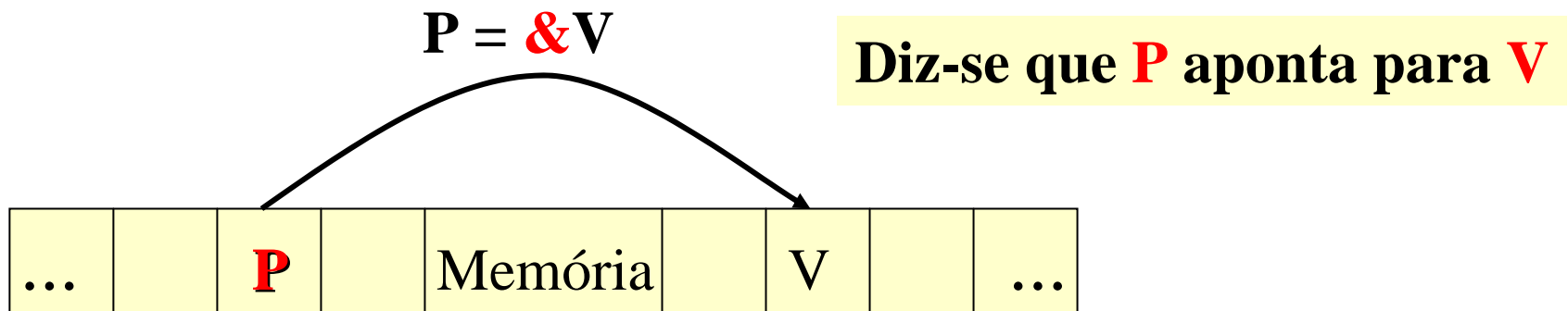
P = endereço da variável **V**

Por que ponteiros são usados ?

- Possibilitar que funções modifiquem os argumentos que recebem
- Manipular matrizes e strings de forma mais facilmente através da movimentação de ponteiros para elas (ou parte delas) - útil para passar matrizes como parâmetro
- Criar estruturas de dados mais complexas, como listas encadeadas, árvores binárias etc.
- Código mais eficiente (compilam mais rápido)

Operador &

- Fornece o endereço de um objeto



- Só se aplica a objetos na memória: variáveis e matrizes
- Não pode ser aplicado a expressões ou constantes

- Ex.: $x = \&3;$

ERRADO

Operador de Indireção *

- Quando aplicado a um ponteiro, acessa o objeto apontado por ele

```
int x=1, y=2, z[3];
```

```
int *ip;      /* ip é um ponteiro para int */
```

```
ip=&x;        /*ip aponta para o endereço de x*/
```

```
y=*ip;        /*y = valor de ip (ou seja, 1)*/
```

```
*ip=0;
```

```
ip=&z[0];
```

x	0
y	1
ip	
Z[0]	?
Z[1]	?
Z[2]	?
⋮	⋮

Ponteiros

- Suponha o código abaixo:

`x=5;`

`y=x;`

`x=x+1;`

- Como atualizar o valor de Y sempre que X for alterado para que *sempre* $X = Y$???
 - » Solução 1: Qualquer alteração em X deve ser seguida pela linha $Y=X$
 - » Solução 2: Ponteiros !!

Ponteiros

```
int x, *y;
```

```
x = 5;
```

```
y = &x;
```

```
x++;
```

```
printf ("X = %d, Y= %d\n", x, *y);
```



**Y mantém o mesmo
valor de X !!**

Passando endereços para uma função

- Como uma função pode alterar variáveis da função chamadora ?
 - 1) função chamadora passa os endereços dos valores que devem ser modificados
 - 2) função chamada deve declarar os endereços recebidos como ponteiros

Passando endereços para uma função

```
main( )  
{  
    int x, y;  
    x=0;  
    y=0;  
    altera2(&x,&y);  
    printf("O 1o. é %d, o 2o. é %d.", x, y);  
}
```

```
altera2(int *px, int *py)  
{  
    *px = 3;  
    *py = 5;  
}
```

***px e *py são do tipo int**

px e py contêm endereços de variáveis do tipo int

Passando endereços para uma função

Chamada por referência

```
main( )
{
    int x, y;
    x=0;
    y=0;
    altera2(&x,&y);
    printf("1º é %d, 2º é %d.",x, y);
}
```

```
altera2(int *px, int *py)
```

```
{
    *px = 3;
    *py = 5;
}
```

Altera os valores
de x e y

Chamada por valor

```
main( )
{
    int x, y;
    x=0;
    y=0;
    altera2(x,y);
    printf("1º é %d, 2º é %d.",x, y);
}
```

```
altera2(int px, int py)
```

```
{
    px = 3;
    py = 5;
}
```

Não altera os
valores de x e y

Passando endereços para uma função

Escreva um programa que aplica a função **exponencial_2** a uma variável inteira e imprime o resultado da aplicação fornecido pelo quadrado do valor da variável.

Obs1: a função **exponencial_2** deve ser do tipo void e eleva um número ao quadrado

Obs2: o resultado deve ser armazenado na própria variável inteira passada como parâmetro para **exponencial_2**

Passando endereços para uma função

```
main( )
{
    int x;
    x=2;
    exponencial_2(&x);
    printf("o resultado é %d.", x);
}

exponencial_2(int *px)
{
    *px = (*px)*(*px);
}
```

```
main( )
{
    int x;
    int *px;
    x = 2;
    px = &x;
    *px = (*px)*(*px);
    printf("o resultado é %d.", x);
}
```

Tem o mesmo efeito
sem usar a função

Operações com ponteiros

```
main( )
```

```
{
```

```
    int x=5, y=6;
```

```
    int *px, *py;
```

```
    px = &x;
```

```
    py = &y;
```

```
    if (px<py) printf("py-px = %u\n",py-px);
```

```
    else      printf("px-py = %u\n",px-py);
```

```
    printf("px = %u, *px = %d, &px = %u\n", px, *px, &px);
```

```
    printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
```

```
    px++;
```

```
    printf("px = %u, *px = %d, &px = %u\n", px, *px, &px);
```

```
    py=px+3;
```

```
    printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
```

```
    printf("py-px = %u\n",py-px);
```

```
}
```


Operações com ponteiros

py - px = 1

px = 65488, *px = 5; &px = 65492

py = 65490, *py = 6; &py = 65494

px = 65490, *px = 6; &px = 65492

py = 65496, *py = ? &py = 65494

py - px = 3

Testes relacionais \geq , \leq , $<$, $>$, $==$, são aceitos em ponteiros

A diferença entre dois ponteiros será dada na unidade do **tipo de dado apontado**

Operações com ponteiros

- O incremento de um ponteiro acarreta na movimentação do mesmo para o próximo valor do *tipo apontado*
 - Ex: Se *px* é um ponteiro para int com valor 3000, depois de executada a instrução *px++*, o valor de *px* será 3002 e não 3001 !!!
 - » Obviamente, o deslocamento varia de compilador para compilador dependendo do número de bytes adotado para o referido tipo

Ponteiros e Matrizes

- Em C existe um relacionamento muito forte entre ponteiros e matrizes
 - O compilador transforma toda matriz em ponteiros, pois a maioria dos computadores é capaz de manipular ponteiros e não matrizes
 - Qualquer operação que possa ser feita com índices de uma matriz pode ser feita com ponteiros
 - Ponteiros e matrizes acessam a memória de forma idêntica
 - O nome de uma matriz é um endereço, ou seja, um ponteiro

Ponteiros e Matrizes

Versão com Matriz

```
main( )
{
    int nums[ ] = {1, 4, 8};
    int cont;

    for(cont=0; cont < 3; cont++)
        printf("%d\n,nums[cont]);
}
```

Versão com Ponteiro

```
main( )
{
    int nums[ ] = {1, 4, 8};
    int cont;

    for(cont=0; cont < 3; cont++)
        printf("%d\n",*(nums + cont));
}
```

Endereço inicial da matriz

Deslocamento

Ponteiros e Matrizes

Observe a diferença para.....

```
main( )
{
    int nums[ ] = {1, 4, 8};
    int cont;

    for(cont=0; cont < 3; cont++)
        printf(“%d\n”, (nums + cont));
}
```

Sem o * !!

Ponteiros e Matrizes

- Observação sobre o tamanho da matriz:
 - Comando *sizeof*

```
main()
{
    static int num[ ]={1,2,3};

    printf ("Tamanho = %d\n", sizeof(num));
    printf ("Numero de elementos = %d\n", sizeof(num)/sizeof(int));
}
```

Observe o resultado !!

Observe o resultado !!

O que representam ?

Ponteiros e Matrizes

- Escreva um programa que lê um conjunto de, no máximo, 40 notas, armazena-as em uma matriz e, por fim, imprime a média das notas.
 - Obs: utilize ponteiros para manipular a matriz
 - O programa pára de pedir as notas e prossegue com o cálculo da média quando o usuário entra com uma nota < 0

Ponteiros e Matrizes

```
main( )
{
    float notas[40], soma=0;
    int cont=0;
    do {
        printf("Digite a nota do aluno %d: ", cont);
        scanf("%f",&*(notas+cont));
        if(*(notas+cont) > -1)
            soma += *(notas+cont);
    } while(*(notas+cont++) > 0);
    printf("Média das notas: %.2f", soma/(cont-1));
}
```

Endereço !!

**++ refere-se
ao cont !!**

Ponteiros e Matrizes

**Mesmo
Resultado !!!**

```
main( )
{
    float notas[40], soma=0;
    int cont=0;
    do {
        printf("Digite a nota do aluno %d: ", cont);
        scanf("%f", notas+cont);
        if(*(notas+cont) > -1)
            soma += *(notas+cont);
    } while(*(notas+cont++) > 0);

    printf("Média das notas: %.2f", soma/(cont-1));
}
```

Ponteiros e Matrizes

**Observe
agora.....**

```
main( )
{
    float notas[40], soma=0;
    int cont=0;
    do {
        printf("Digite a nota do aluno %d: ", cont);
        scanf("%f", &*(notas+cont));
        printf ("%d\n", (notas+cont));
        if(*(notas+cont) > -1)
            soma += *(notas+cont);
    } while(*(notas+cont++) > 0);

    printf("Média das notas: %.2f", soma/(cont-1));
}
```

Ponteiros e Matrizes

Será que existe alguma maneira de simplificar a expressão
`while(*(notas+cont++)>0)` ?

`while (*(notas++) > 0)`

Errado !! “notas” é um
ponteiro constante ! É o
endereço da matriz notas
e não pode ser trocado
durante a execução do
programa !

Apenas um ponteiro variável pode ser alterado

Ponteiros e Matrizes

- **Vamos re-escrever o programa anterior com um ponteiro variável....**

```
main( )
{
    float notas[40], soma=0;
    int cont=0; float *ptr;
    ptr = notas;
    do {
        printf("Digite a nota do aluno %d: ", cont++);
        scanf("%f",ptr);
        if(*ptr > -1)
            soma += *ptr;
    } while(*(ptr++) > 0);
    printf("Média das notas: %.2f", soma/(cont-1));
}
```

Passando Matrizes como Parâmetro

Somando uma constante aos elementos de uma matriz

```
#define TAM 5
main( )
{
    int matriz[TAM]={3,5,7,9,11};
    int c=10;
    int j;

    adcons(matriz, TAM, c);
    for(j=0; j<TAM; j++)
        printf("%d,*(matriz+j));
    }

adcons(int *ptr, int num, int cons)
{
    int k;
    for(k=0; k<num; k++)
        *ptr = *(ptr++) + cons;
    }
```

Ponteiro e String

- Um string é um array de caracteres
- Exemplo:

```
main( )  
{  
    char c='t';  
    char nome[10]="teste";  
  
    printf ("Texto1 = %c\n", c);  
    printf ("Texto2 = %s\n", nome);  
}
```

Observe a
diferença
na notação !

Ponteiro e String

- Exemplo 2:
 - Modifique o programa anterior para...

```
main( )  
{  
    char c='t';  
    char nome[10]="teste de texto";  
  
    printf ("Texto1 = %c\n", c);  
    printf ("Texto2 = %s\n", nome);  
}
```

O que
acontece ?

Ponteiro e String

- Exemplo 3:
 - Modifique novamente para...

```
main( )  
{  
    char c='t';  
    char nome[10];  
    scanf("%s", nome);  
    printf("%c\n", c);  
    printf ("%s\n", nome);  
}
```

Digite:
cddvfvfddfbgbgghnh
(qualquer coisa com
mais de 10 caracteres)

Inicialização de string através de ponteiros

```
main( )  
{  
    char *salute="saudacoes";  
    char nome[8];  
  
    puts("Digite seu nome");  
    gets(nome);  
    puts(salute);  
}
```

Ponteiro variável

puts(++salute); → saudações (sem S)

```
main( )  
{  
    char salute[ ]="saudacoes";  
    char nome[8];  
  
    puts("Digite seu nome");  
    gets(nome);  
    puts(salute);  
}
```

Ponteiro constante

puts(++salute); → **ERRO**

Inicialização de uma matriz de ponteiros para string

```
main( )
{
    int cont;
    int entra=0;
    char nome[40];
    static char *list[5]= {"Carlos","Ana","Pedro","Andre", "Silvia" };
    printf ("Digite seu nome: ");
    gets(nome);
    for (d=0; d < 5; d++)
        if (strcmp(list[d],nome) == 0)
            entra = 1;
    if (entra == 1)
        printf ("Voce pode entrar");
    else
        printf ("Voce nao pode entrar");
}
```

Inicialização de uma matriz de ponteiros para string

```
main( )
{
    int cont;
    int entra=0;
    char nome[40];
    static char list[5][7]=
        { "Carlos", "Ana", "Pedro", "Andre", "Silvia" };
    printf ("Digite seu nome:");
    gets(nome);
    for (cont=0; cont<5; cont++)
        if (strcmp(list[cont],nome) == 0)
            entra=1;
    if (entra == 1)
        printf ("Você pode entrar.");
    else
        printf ("Você não pode entrar.");
}
```

**Versão usando array
Cuidado com as
dimensões da matriz !**

Inicialização de uma matriz de ponteiros para string

```
main( )
{
    int cont;
    int entra=0;
    char nome[40];
    static char list[5][6]=
        { "Carlos", "Ana", "Pedro", "Andre", "Silvia" };
    printf ("Digite seu nome:");
    gets(nome);
    for (cont=0; cont<5; cont++)
        if (strcmp(list[cont],nome) == 0)
            entra=1;
    if (entra == 1)
        printf ("Você pode entrar.");
    else
        printf ("Você não pode entrar.");
}
```

Modifique o número de colunas para 6 e teste... Veja se Carlos pode entrar e se CarlosAna pode.....

Inicialização de uma matriz de ponteiros para string

Versão Array

	0	1	2	3	4	5	6	
list[0] →	C	a	r	l	o	s	\0	
list[1] →	A	n	a	\0				
list[2] →	P	e	d	r	o	\0		← Desperdício
list[3] →	A	n	d	r	e	\0		
list[4] →	S	i	l	v	i	a	\0	

Versão ponteiro

list[0] →	C	a	r	l	o	s	\0
list[1] →	A	n	a	\0			
list[2] →	P	e	d	r	o	\0	
list[3] →	A	n	d	r	e	\0	
list[4] →	S	i	l	v	i	a	\0

Escreva um programa para colocar a lista de nomes apontadas pelo array em ordem alfabética

Lista Original

list[0]	→	C	a	r	l	o	s	\0
list[1]	→	A	n	a	\0			
list[2]	→	P	e	d	r	o	\0	
list[3]	→	A	n	d	r	e	\0	
list[4]	→	S	i	l	v	i	a	\0

Lista Ordenada

list[0]	↗	A	n	a	\0			
list[1]	↘	A	n	d	r	e	\0	
list[2]	↗	C	a	r	l	o	s	\0
list[3]	↘	P	e	d	r	o	\0	
list[4]	→	S	i	l	v	i	a	\0

```
#include "stdio.h"
#include "conio.h"
#include "string.h"
main( )
{
    int cont1, cont2;
    char *temp;
    static char *list[5] = {"Carlos", "Ana", "Pedro", "Andre", "Silvia"};

    printf("\nLista Original:\n\n");
    for(cont1=0; cont1<5; cont1++)
        printf("Nome %d: %s\n",cont1,list[cont1]);

    for(cont1=0; cont1<5; cont1++)
        for(cont2=cont1+1; cont2 < 5; cont2++)
            if (strcmp(list[cont1] , list[cont2]) > 0){
                temp      = list[cont2];
                list[cont2] = list[cont1];
                list[cont1] = temp;
            }

    printf("\nLista Ordenada:\n\n");
    for(cont1=0; cont1 < 5; cont1++)
        printf("Nome %d: %s\n",cont1,list[cont1]);
}
```

Alocando a Memória: Função malloc()

- Aloca somente a memória necessária quando desejarmos
- A função malloc() toma um inteiro sem sinal como argumento, representando a quantidade em bytes de memória requerida
- Retorna um ponteiro para o primeiro byte de memória que foi alocado

Alocando a Memória: Função malloc()

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>
main()
{
    char *str;
    /* allocate memory for string */
    if ((str = (char *) malloc(10)) == NULL)
    {
        printf("Not enough memory to allocate buffer\n");
        exit(1); /* terminate program if out of memory */
    }
    /* copy "Hello" into string */
    strcpy(str, "Hello");
    /* display string */
    printf("String is %s\n", str);
    /* free memory */
    free(str);
}
```

Liberando a Memória: Função free()

- Complemento de malloc()
- Aceita como argumento um ponteiro para uma área de memória previamente alocada por malloc() e então libera esta área para possível utilização futura
- É importante liberar a memória alocada após o seu uso, pois esta técnica pode resultar numa quantidade grande de memória reutilizável

Alocando uma Matriz na Memória com Malloc

- Deve-se usar um ponteiro para ponteiro
 - `int **v`
- Para alocar uma matriz na memória utilizando a função malloc, é preciso fazer a alocação de todas as linhas e, em seguida, dos elementos de cada linha
- Da mesma forma, a liberação da memória é feita em partes

Alocando uma Matriz na Memória com Malloc

```
main( )
{
    int **v;
    v=(int**) malloc(10*sizeof(int));
    for (int i=0; i<=9; i++)
        v[i] = (int*) malloc(5*sizeof(int));
    for (int i=0; i<=9; i++)
        for (int j=0; j<=4; j++)
            v[i][j] = i+j;
    for (int i=0; i<=9; i++)
        for (int j=0; j<=4; j++)
            printf ("%d\n", v[i][j]);
    for (int i=0; i<=9; i++)
        free(v[i]);
    free(v);
}
```

Alocando uma matriz
10x5 de inteiros

Liberando os elementos
da matriz

Alocando uma Matriz na Memória com Malloc

- Observe as posições de memória sendo alocadas fazendo a modificação abaixo...

```
for (int i=0; i<=9; i++)  
    for (int j=0; j<=4; j++)  
    {  
        v[i][j] = i+j;  
        printf ("Posicao = %u, Valor = %d\n", &v[i][j], v[i][j]);  
        getche();  
    }
```

Passando uma Matriz como Saída de uma Função

- A função abaixo recebe uma matriz e soma um valor qualquer aos elementos dessa matriz e devolve outra matriz de saída....

```
int** add(int **ptr)
{
    int i, j, c = 20;
    for (i=0; i<tam; i++)
        for (j=0; j<tam; j++)
            ptr[i][j] = ptr[i][j] + c;
    return(ptr);
}
```

Passando uma Matriz como Saída de uma Função

```
main( )
{ int **mat, **mat2, i, j;
  mat = (int**) malloc(tam*sizeof(int));
  for (i=0; i<tam; i++)    mat[i] = (int*) malloc(tam*sizeof(int));
  mat2 = (int**) malloc(tam*sizeof(int));
  for (i=0; i<tam; i++)    mat2[i] = (int*) malloc(tam*sizeof(int));
  for (i=0; i<tam; i++)
    for (j=0; j<tam; j++)
      mat[i][j] = 0;
  mat2 = add(mat);
  for (i=0; i<tam; i++)
    for (j=0; j<tam; j++)
      printf ("%d\n", mat[i][j]);
  for (i=0; i<tam; i++)
    { free (mat[i]); free (mat2[i]); }
}
```

Programa
Principal

Linguagens de Programação I

Linguagem C

Estruturas

Carlos Mello
Ricardo Massa



Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

Estruturas

- Agrupa conjunto de tipos de dados distintos sob um único nome
- Chamadas de Registros

Cadastro Pessoal

string	Nome
string	Endereço
inteiro	Telefone
inteiro	Idade
inteiro	Data de Nascimento
float	Peso
float	Altura

```
struct cadastro_pessoal {  
    char *nome;  
    char *endereço;  
    int    telefone;  
    int    idade;  
    int    nascimento;  
    float  peso;  
    float  altura;  
}
```

Estruturas


■ Um pequeno exemplo

```
main( )
{
    struct facil {
        int num;
        char ch;
    };

    struct facil var; /*declara variável var do tipo facil */
    var.num = 2;
    var.ch  = 'Z';
    printf("var.num = %d, var.ch = %c\n",var.num, var.ch);
}
```

Estruturas

■ Outra forma...

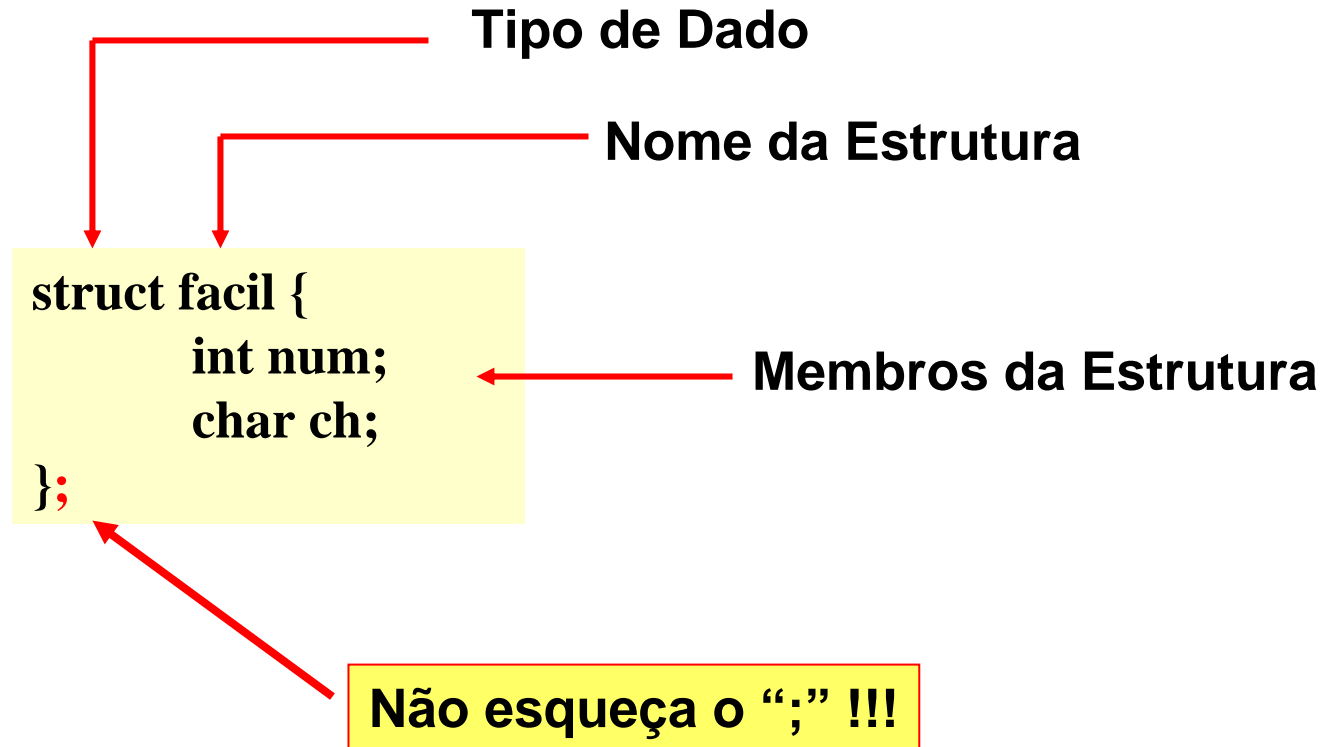


```
main( )
{
    struct facil {
        int num;
        char ch;
    } var; /*declara variável var do tipo facil */

    var.num = 2;
    var.ch   = 'Z';
    printf("var.num = %d, var.ch = %c\n",var.num, var.ch");
}
```

Estruturas

■ Análise



Estruturas

Mais de uma
variável

```
main( )
{
    struct facil {
        int num;
        char ch;
    };
    struct facil var1; /*declara variável var do tipo facil */
    struct facil var2; /*declara variável var do tipo facil */

    var1.num = 2;  var1.ch  = 'Z';
    var2.num = 3;  var2.ch  = 'B';
    printf("var1.num = %d, var1.ch = %c\n",var1.num, var1.ch");
    printf("var2.num = %d, var2.ch = %c\n",var2.num, var2.ch");
}
```

Estruturas

Ou :

```
main( )
{
    struct facil {
        int num;
        char ch;
    } var1, var2;

    var1.num = 2;  var1.ch  = 'Z';
    var2.num = 3;  var2.ch  = 'B';
    printf("var1.num = %d, var1.ch = %c\n", var1.num, var1.ch);
    printf("var2.num = %d, var2.ch = %c\n", var2.num, var2.ch);
}
```

Estruturas

- Pode-se definir uma estrutura sem um nome...

```
struct {  
    int num;  
    char ch;  
} var1, var2;
```

Estruturas

- Criando uma Lista de Livros
 - Passo inicial: 2 Livros

```
struct livro {  
    char titulo[40];  
    int  regnum;  
};
```

```
struct livro livro1 = {"Treinamento em Linguagem C - I", 124};  
struct livro livro2 = {"Treinamento em Linguagem C - II", 125};
```


Atribuições entre estruturas

- Na versão original do C, definida por Kernighan e Ritchie, era impossível atribuir o valor de uma variável estrutura a outra do mesmo tipo usando uma simples expressão de atribuição.
- Nas versões modernas de C, esta forma de atribuição já é possível

Atribuições entre estruturas

```
struct livro {  
    char titulo[40];  
    int  regnum;  
};
```

```
struct livro livro1 = {"Treinamento em Linguagem C - I", 124};  
struct livro livro2 = {"Treinamento em Linguagem C - II", 125};
```

```
livro2 = livro1;
```

Estruturas Aninhadas

- Exatamente como é possível ter matrizes de matrizes, pode-se criar estruturas que contêm outras estruturas.

```
struct professor {  
    char nome[50];    char disciplina[20];  
    int  carga_horaria;  
};  
struct aluno {  
    char nome[50];    int matricula;  
};  
struct cadastro_escolar {  
    struct professor docentes[100];  
    struct aluno     discente[10000];  
};
```

Passando estruturas para funções

- As novas versões de C permitem que uma função passe ou retorne uma estrutura completa para outra função
- Exemplo: Criar uma função para obter dos usuários dados sobre os livros

Passando estruturas para funções

```
struct livro {  
    char titulo[40];  
    int regnum;  
};
```

Global

```
main( )  
{  
    struct livro livro1;  
    struct livro livro2;  
    struct livro novonome( );  
    livro1 = novonome( );  
    livro2 = novonome( );  
    list(livro1);  
    list(livro2);  
}
```

```
struct livro novonome( )
```

```
{  
    char numstr[81];  
    struct livro livr;  
    printf("\nDigite título:");  
    gets(livr.titulo);  
    printf("Digite registro:");  
    gets(numstr);  
    livr.regnum=atoi(numstr);  
    return(livr);  
}
```

```
void list(struct livro livr)
```

```
{  
    printf("\nLivro:\n");  
    printf("  Título: %s\n", livr.titulo);  
    printf("  No do registro: %3d\n",livr.regnum);  
}
```

Passando estruturas para funções

- No programa anterior, poderíamos ter a função novonome como uma função normal (sem ser uma estrutura) ?

```
main( )
{
    struct livro livro1;
    struct livro livro2;
    livro1 = novonome( );
    livro2 = novonome( );
    list(livro1);
    list(livro2);
}
```

```
novonome( )
{
    char numstr[81];
    struct livro livr;
    printf("\nDigite título:");
    gets(livr.titulo);
    printf("Digite registro:");
    gets(numstr);
    livr.regnum=atoi(numstr);
    return(livr);
}
```

OK

Ponteiros para Estruturas

```
struct lista {  
    char titulo[30];  
    char autor[30];  
    int regnum;  
    double preco;  
};  
main()  
{
```

Endereço 1: 404 2: 474

Ponteiro 1: 404 2: 474

ptrl->preço: R\$ 70.5 (*ptrl).preço: R\$ 70.5

ptrl->título: C++ ptrl->autor: Alexandre

```
    static struct lista livro[2] =  
        { { "C", "Carlos", 102, 70.5 },  
          { "C++", "Alexandre", 321, 63.25} };  
    struct lista *ptrl; /* ponteiro para estrutura */  
    printf("Endereço 1: %u 2: %u\n", &livro[0], &livro[1]);  
    ptrl = &livro[0];  
    printf("Ponteiro 1: %u 2: %u\n", ptrl, ptrl+1);  
    printf("ptrl->preço: R$.%2f (*ptrl).preço: R$.%2f\n", ptrl->preço, (*ptrl).preço);  
  
    ptrl++; /* aponta para a próxima estrutura */  
    printf("ptrl->título: %s ptrl->autor: %s\n", ptrl->título, ptrl->autor);  
}
```

Ponteiros para Estruturas

- `struct lista *ptrl;`
 - O ponteiro **ptrl** pode apontar para qualquer estrutura do tipo **lista**
- `ptrl = &livro[0];`
 - **ptrl** aponta para **livro[0]**

Ponteiros para Estruturas

Acessando os membros através do ponteiro

- Normalmente, os membros de uma estrutura são acessados usando seu nome seguido do operador ponto
- O mesmo acontece com ponteiros ?
- Se **struct lista *ptrl**, poderíamos escrever **ptrl.preço** ?
- NÃO !!!

Ponteiros para Estruturas

Acessando os membros através do ponteiro

- Correto:

- `ptrl->preço`

- ou

- `(*ptrl).preço`

- » Os parênteses são obrigatórios !!!

- » Sem eles, a estrutura seria lida como `*(ptrl.preço)` que geraria um erro !

Linguagens de Programação I

Linguagem C

Unões

Carlos Mello
Ricardo Massa

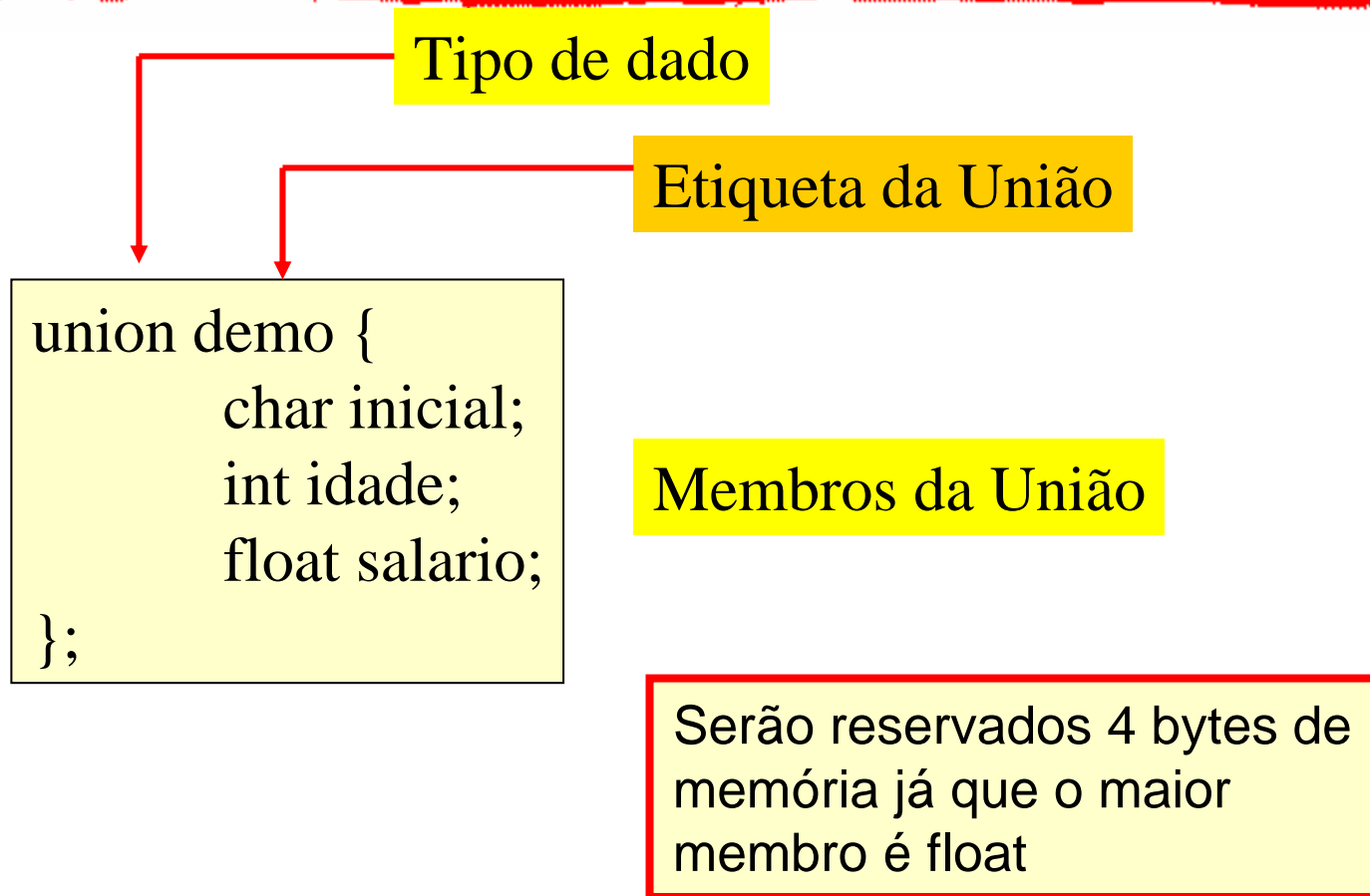


Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

União

- Define uma mesma seção da memória contendo um tipo de variável em uma ocasião e outro tipo em outra ocasião
- Quando uma união é declarada, o espaço de memória alocado para ela será igual ao do maior tipo de dado

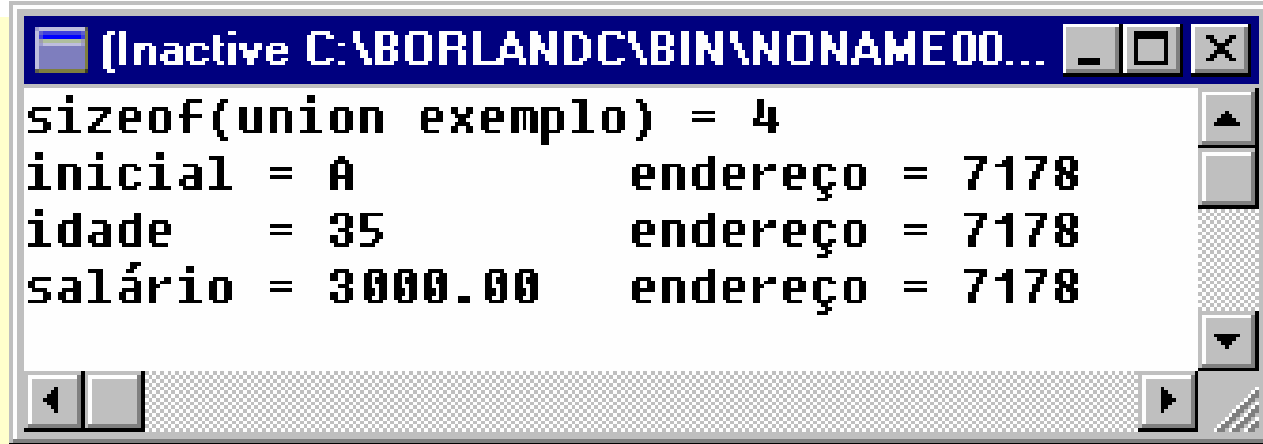
União



União

```
main( )
{
    union exemplo {
        char inicial;
        int idade;
        float salário;
    };

    union exemplo pessoal;
    printf("sizeof(union exemplo) = %d\n", sizeof(union exemplo));
    pessoal.inicial = 'A';
    printf("inicial = %c    endereço = %u\n", pessoal.inicial, &pessoal.inicial);
    pessoal.idade = 35;
    printf("idade = %d    endereço = %u\n", pessoal.idade, &pessoal.idade);
    pessoal.salário = 3000.0;
    printf("salário = %.2f    endereço = %u\n", pessoal.salário, &pessoal.salário);
}
```



```
(Inactive C:\BORLANDC\BIN\NONAME00...
sizeof(union exemplo) = 4
inicial = A          endereço = 7178
idade   = 35         endereço = 7178
salário = 3000.00    endereço = 7178
```

União

- Por que usar Union ?
 - Conveniente quando queremos que um mesmo dado seja tratado com tipos diferentes
 - Preservar memória

Linguagens de Programação I

Linguagem C

Arquivos

Carlos Mello
Ricardo Massa



Engenharia da Computação
Escola Politécnica de Pernambuco - UPE

Operações com Arquivos em Disco

- Existem dois tipos de arquivo: texto e binário
- Arquivo Texto:
 - Interpretado como seqüência de caracteres agrupados em linhas
 - Linhas são separadas por um único caractere denominado LF
 - Em DOS as linhas são separadas por CR/LF
 - O compilador C converte o par CR/LF em LF quando o arquivo é lido e LF em CR/LF quando ele é gravado
 - DOS fornece uma indicação de fim de arquivo ao programa quando ele tenta ler alguma informação após último caractere

Operações com Arquivos em Disco

■ Arquivo binário:

- Nenhuma conversão é feita, qualquer caractere é lido ou gravado sem alteração
- Nenhuma indicação de fim de arquivo é reconhecida
- Enquanto no modo texto os números são armazenados como cadeias de caracteres, no modo binário eles são armazenados da mesma forma que ele é representado na memória

Abrindo Arquivos

- `FILE *fptr; /* ponteiro para arquivo */`
- `fptr = fopen("arqtext.txt", "w");`

nome do arquivo

Tipo de abertura

- “**r**” Abrir arquivo texto para leitura. O arquivo deve estar presente no disco
- “**w**” Abrir arquivo texto para gravação. Se o arquivo existir ele será destruído e reinicializado. Se não existir, será criado
- “**a**” Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo existente, ou cria um novo

Abrindo arquivos

- `FILE *fptr; /* ponteiro para arquivo */`
- `fptr = fopen("arqtext.txt", "w");`

nome do
arquivo

Tipo de abertura

- “**r+**” Abrir arquivo texto para leitura e gravação. O arquivo deve existir e pode ser **atualizado**.
- “**w+**” Abrir arquivo texto para leitura e gravação. Se o arquivo existir ele será **destruído e reinicializado**. Se não existir, será criado.
- “**a+**” Abrir um arquivo texto para atualização e para adicionar dados no fim do arquivo existente, ou cria um novo

Abrindo arquivos

- `FILE *fptr; /* ponteiro para arquivo */`
- `fptr = fopen("arqtext.txt", "w");`

nome do arquivo

Tipo de abertura

- “rb”** Abrir arquivo binário para leitura. O arquivo deve estar presente no disco
- “wb”** Abrir arquivo binário para gravação. Se o arquivo existir ele será destruído e reinicializado. Se não existir, será criado
- “ab”** Abrir um arquivo binário para gravação. Os dados serão adicionados no fim do arquivo existente, ou cria um novo

Abrindo arquivos

- `FILE *fptr; /* ponteiro para arquivo */`
- `fptr = fopen("arqtext.txt", "w");`

nome do
arquivo

Tipo de abertura

- “**rb+**” Abrir arquivo binário para leitura e gravação. O arquivo deve existir e pode ser **atualizado**.
- “**wb+**” Abrir arquivo binário para leitura e gravação. Se o arquivo existir ele será **destruído e reinicializado**. Se não existir, será criado.
- “**ab+**” Abrir um arquivo binário para atualização e para adicionar dados no fim do arquivo existente, ou cria um novo

Fechando arquivos

- `FILE *fptr; /* ponteiro para arquivo */`
- `fptr = fopen("arqtext.txt", "w");`
- `fclose(fptr);`

Escrevendo em arquivos

```
FILE *fptr; /* ponteiro para arquivo */  
char ch;  
fptr = fopen("arqtext.txt", "w");  
while((ch=getche( )) != '\r')  
    fputc(ch, fptr);  
fclose(fptr);
```


Lendo arquivos

```
FILE *fptr; /* ponteiro para arquivo */  
char ch;  
fptr = fopen("arqtext.txt", "r");  
while((ch=fgetc(fptr)) != EOF)  
    printf("%c",ch);  
fclose(fptr);
```

Cuidados ao abrir arquivos

- A operação para abertura de arquivos pode falhar !!!
 - Falta de espaço em disco
 - Arquivo ainda não criado
- Se o arquivo não puder ser aberto, a função `fopen()` retorna o valor `NULL`

Cuidados ao abrir arquivos

```
FILE *fptr; /* ponteiro para arquivo */
char ch;
if((fptr = fopen("arqtext.txt", "r") == NULL)
{
    printf("Não foi possível abrir o arquivo arqtext.txt");
    exit( );
}
while((ch=fgetc(fptr)) != EOF)
printf("%c",ch);
fclose(fptr);
```

Exercício

- Escreva um programa que grava um texto qualquer em um arquivo e, em seguida, conta o número de caracteres do arquivo
 - Observe o tamanho do arquivo e o número de caracteres contados
 - » Há diferença ? Por quê ?

Gravando um arquivo linha a linha

```
FILE *fptr; /* ponteiro para arquivo */
char string[81];
if((fptr = fopen("arqtext.txt", "w") == NULL) {
    printf("Não foi possível abrir o arquivo arqtext.txt");
    exit( );
}
while(strlen(gets(string)) > 0) {
    fputs(string,fptr);
    fputs("\n",fptr);
}
fclose(fptr);
```

Lendo um arquivo linha a linha

```
FILE *fptr; /* ponteiro para arquivo */
char string[81];
if((fptr = fopen("arqtext.txt", "r") == NULL) {
    printf("Não foi possível abrir o arquivo arqtext.txt");
    exit( );
}
while(fgets(string,80,fptr)) != NULL) {
    printf("%s",string);
}
fclose(fptr);
```

Gravando um arquivo de maneira formatada

É possível usar *fscanf* para ler dados do disco.

fscanf é similar à função *scanf*, exceto que, como *fprint*, um ponteiro para **FILE** seja incluído como primeiro argumento

```
FILE *fptr; /* ponteiro para arquivo */
char título[30];
int regnum;
double preço;
if((fptr = fopen("arqtext.txt", "w") == NULL) {
    printf("Não foi possível abrir o arquivo arqtext.txt");
    exit( );
}
do {
    printf("\nDigite título, registro e preço");
    scanf("%s %d %f", título, &regnum, &preço);
    fprintf(fptr, "%s %d %f", título, regnum, preço);
} while(strlen(título) > 1);
fclose(fptr);
```

Exercício

- Escreva um programa que grave o nome, 1a nota, 2a nota e média dos alunos de uma turma em um arquivo. Após a gravação do arquivo, ele deve ser novamente aberto para listar os alunos com suas respectivas notas.

Gravando Estruturas com fwrite

```
struct livros { char título[30];  
                int regnum;  
                double preço;  
            } livro;  
char numstr[81];  
FILE *fptr;  
if((fptr = fopen("livros.arq", "wb")) == NULL) {  
    printf("não posso abrir arquivo livros.arq");  
    exit(1); }  
do { printf("\n Digite o título:");  
    gets(livro.título);  
    printf("\n Digite o registro:");  
    gets(numstr);  
    livro.regnum = atoi(numstr);  
    printf("\n Digite o preço:");  
    gets(numstr);  
    livro.preço = atof(numstr);  
    fwrite(&livro, sizeof(struct livros), 1, fptr);  
    printf("\n Adiciona outro livro (s/n) ?");  
} while (getche( ) == 's');  
fclose(fptr);
```

Gravando matrizes com fwrite

```
int tabela[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
FILE *fptr;
```

```
if((fptr = fopen("tabela.arq", "wb")) == NULL) {
```

```
    printf("\n Não posso abrir o arquivo  
tabela.arq");
```

```
    exit(1);
```

```
}
```

```
fwrite(tabela, sizeof(int), 10, fptr);
```

```
fclose(fptr);
```

Lendo Estruturas com fread

```
struct livros { char título[30];
                int regnum;
                double preço;
            } livro;
char numstr[81];
FILE *fptr;
if((fptr = fopen("livros.arq", "rb")) == NULL) {
    printf("não posso abrir arquivo livros.rec");
    exit(1); }
while (fread(&livro, sizeof(struct livros), 1, fptr) == 1) {
    printf("\n Título: %s\n", livro.título);
    printf("\n Registro: %03d\n", livro.regnum);
    printf("\n Preço: %.2f\n", livro.preço);
}
fclose(fptr);
```