

**USP - ICMC - SSC**  
**SSC 0501 - 1o. Semestre 2011**

**Disciplina de**  
**Introdução à Ciência da Computação**  
**ICC 1 - Teoria**

**Prof. Fernando Santos Osório**

**Email: fosorio [at] { icmc. usp. br , gmail. com }**

**Página Pessoal: <http://www.icmc.usp.br/~fosorio/>**

**Web - WIKI ICMC: <http://wiki.icmc.usp.br/index.php/SSC-501>**

**PAE: Daniel Sales (Mestr. CCMC - LRM)**

**Email: dsales [at] icmc.usp.br**

**Monitor: Danilo Alvares da Silva**

**E-mail: danilo [at] grad.icmc.usp.br**

## Linguagem de Programação "C"

### Agenda:

- **Tipos de Dados:**
  - Alocação Estática de Memória
  - Alocação Dinâmica de Memória
- **Endereços e Ponteiros**
  - Alocação de Vetores e Matrizes
  - Alocação de Vetores de Caracteres (Strings)
  - Funções em "C" de alocação de memória: Calloc, Malloc

**Informações Complementares e Atualizadas:**

**Consulte REGULARMENTE**

**O material disponível na COTEIA**

## Alocação de Memória

### Alocação Estática:

- Espaço de memória é reservado previamente;
- Área de Tamanho FIXO e de Endereço FIXO (estática);
- Declarado e reservado em tempo de compilação  
Exemplo: *variáveis globais* - declaradas fora do main/funções

### Alocação Dinâmica:

- Espaço de memória é alocado em tempo de execução usando as funções “calloc” e “malloc” (dinâmica);
- Áreas de Tamanho VARIÁVEL e de Endereço VARIÁVEL são criadas (reserva memória) e destruídas (libera memória);
- Acesso usualmente através de Endereços e Ponteiros  
Exemplo: *ponteiros+calloc/malloc* ou *variáveis locais*

## Alocação de Memória

### Alocação Estática:

- Variáveis GLOBAIS: Acesso “livre” em qualquer parte do programa, são declaradas fora de qualquer bloco {...}

Exemplo: *variáveis globais* - declaradas fora do main/funções

### Alocação Dinâmica:

- Variáveis LOCAIS: Acesso restrito a uma parte do programa, são declaradas dentro de um bloco {...} logo após o começo do bloco (declaração após o abre chaves “{“ e sendo assim pertencem a um bloco, logo, só poderão ser usadas dentro de deste bloco no qual foram declaradas.

Exemplo: *ponteiros+calloc/malloc* ou *variáveis locais*

### Operadores de Endereço:

- Realizam operações com endereços de memória.

#### SINTAXE:

**&** Obtém o endereço de uma variável. Ponteiros são “tipados”

Exemplo:

```
int num;           /* Variável int */  
int *ptr_num;     /* Ponteiro para uma variável int */  
ptr_num = &num;  /* Obtém o endereço da variável int num */
```

**\*** Acessa o conteúdo de um endereço especificado.

Exemplo:

```
num = *ptr_num;  /* Atribui o valor contido no endereço  
apontado por ptr_num para num */
```

### Operadores de Endereço:

- Realizam operações com endereços de memória.

#### SINTAXE:

**&** Obtém o endereço de uma variável. Ponteiros são “tipados”

Exemplo:

```
char letra;       /* Variável char */  
char *ptr_char;  /* Ponteiro para uma variável char */  
ptr_char = &letra; /* Obtém o endereço da variável char letra */
```

**\*** Acessa o conteúdo de um endereço especificado.

Exemplo:

```
letra = *ptr_char; /* Atribui o valor contido no endereço  
apontado por ptr_char para letra */
```

## Memória: Endereços e Ponteiros

### Revisão de Ponteiros em "C"

#### Operadores de Endereço: Ponteiros

- Realizam operações com endereços de memória.

SINTAXE:

**&** → Obtém o endereço de uma variável. Exemplo:

```
int dado=51;
int *x;
x = &dado;
```

**\*** → Escreve/Lê o conteúdo do endereço especificado. Exemplo:

```
dado1 = *x; ou *x = dado1;
```

	100	
dado	51	
	102	
	103	
	104	
	105	
Variável	Dado	End.

Ponteiros são TIPADOS: Apontam para um determinado tipo de dado

Declaração:

```
{ int *ptr_valint;
  double *ptr_valreal; ...
```

Utilização:

```
*ptr_valint = dado_int; /* escreve */
ptr_valreal = &dado_real; /* aponta */
```

7

Maio 2010

## Memória: Endereços e Ponteiros

### Revisão de Ponteiros em "C"

#### Operadores de Endereço: Ponteiros

Exemplos:

main ()

```
{
  int vetor[10]; /* Vetor de Inteiros: vetor == &(vetor[0]) */
  int dado;
  int *ptr;

  ptr = &dado; /* Ptr obtém o endereço da variável dado */
  *ptr = 130; /* Armazena o valor 130 no endereço, ou seja, na variável dado! */

  ptr = vetor; /* Ptr obtém o endereço inicial do vetor de inteiros */
  *ptr = 130; /* Armazena o valor 130 no vetor[0] */
  ptr++; /* Aponta para o próximo valor do vetor, o próximo inteiro */
  *ptr = 131; /* Armazena o valor 131 no vetor[1] */
  *vetor = 1; /* Armazena o valor 1 no vetor[0], pois vetor aponta p/ &(vetor[0]) */
  *(vetor+5) = 2; /* Armazena o valor 2 no vetor[5] */
}
```

	100	
dado	51	
	102	
	103	
	104	
	105	
Variável	Dado	End.

8

Maio 2010

## Vetores: Ponteiros e Endereços

### Ponteiros na Linguagem “C” :

- Em vez de usarmos variáveis, nomes que “escondem” os endereços de memória onde estão armazenados os dados, podemos acessar diretamente os dados através de seus endereços...

Declaração: `<tipo_variável> *<nome_variável>;`

Uso:

`&<nome_variável>` => Obtém o endereço da variável

`* <nome_variável>` => Obtém o conteúdo apontado pelo ponteiro

Operadores: ++,-- => Podemos realizar operações aritméticas...

**ATENÇÃO:** Declarar um ponteiro não implica em alocar a memória p/o dado!

Exemplo:



```
int a=10;          /* a == 10 */
int *ptr_int;     /* ponteiro */

ptr_int = &a;    /* ptr_int => a */
*ptr_int = 5;    /* a == 5 ! */
```

```
int vetor[10];

&(vetor[0]) == vetor
&(vetor[1]) == vetor+1
&(vetor[2]) == vetor+2 ...
```

## Vetores: Ponteiros e Endereços

### Tipos de Dados em “C” : Vetores

- Vetores numéricos:

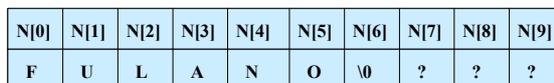
```
int Hora[24];      => Hora[0] .. Hora[23] com valores do tipo "int"
double Notas[10]; => Notas[0] .. Notas[9] com valores do tipo "double"
Notas[0] = 10.0;
```



- Vetores de caracteres:

```
char Letras[26];  => Letras[0] .. Letras[25] com valores do tip "char"
Letras[0] = 'a'; Letras[25] = 'z';
```

```
char Nome[10];    => Nome[0] .. Nome[9] onde uma posição é reservada para a
                  marca de fim da string de nome! (Marca = '\0')
strcpy(Nome, "123456789"); => O Nome não deve ter mais de 9 caracteres, pois o décimo é o '\0'
                  Strings são manipuladas através de rotinas especiais:
                  strcpy, strlen, strcmp, sprintf, sscanf, ... #include <string.h>
```



## Vetores: Ponteiros e Endereços

Tem que reservar (alocar) espaço para guardar os dados!

### Tipos de Dados em "C" : Vetores

- Vetores numéricos:

```
int Hora[24];      => Hora[0] .. Hora[23] com valores do tipo "int"
double Notas[10]; => Notas[0] .. Notas[9] com valores do tipo "double"
Notas[0] = 10.0;
```

```
double *Notas;
*Notas=10.0
```

N[0]	N[1]	N[2]	N[3]	N[4]	N[5]	N[6]	N[7]	N[8]	N[9]
------	------	------	------	------	------	------	------	------	------

- Vetores de caracteres:

```
char Letras[26];      => Letras[0] .. Letras[25] com valores do tip "char"
Letras[0] = 'a'; Letras[25] = 'z';

char Nome[10];        => Nome[0] .. Nome[9] onde uma posição é reservada para
                        marca de fim da string de nome! (Marca = '\0')
strcpy(Nome, "123456789"); => O Nome não deve ter mais de 9 caracteres, pois o décimo é o '\0'
                        Strings são manipuladas através de rotinas especiais:
                        strcpy, strlen, strcmp, sprintf, sscanf, ... #include <string.h>
```

```
char *Nome;
*Nome='F';
*(Nome+1)='U'
```

N[0]	N[1]	N[2]	N[3]	N[4]	N[5]	N[6]	N[7]	N[8]	N[9]
F	U	L	A	N	O	\0	?	?	?

## Vetores: Ponteiros e Endereços

### Ponteiros na Linguagem "C" :

- Quando usamos a rotina scanf já estamos usando ponteiros...

```
scanf ("%d", &var_int); /* Endereço de var_int */
scanf ("%s", var_string); /* Vetor de char */
```

- Alocação de memória:

```
calloc - Aloca memória, zerando os dados
malloc - Aloca memória, sem inicializar os dados
free - Libera um bloco de memória alocada previamente
```

```
void *calloc ( <quantidade_elementos>, <tamanho_elemento> )
```

```
Exemplo: tabela_inteiros = calloc( 10, sizeof ( int ) );
```

```
void *malloc( <quantidade_elementos>, <tamanho_elemento> )
```

```
Exemplo: tabela_inteiros = malloc( 10*sizeof ( int ) );
```

```
void free ( void *ponteiro)
```

```
Exemplo: free ( tabela_inteiros );
```

## Vetores: Ponteiros e Endereços

### Exemplos:

```
int *ptr_int;  
double *ptr_pf;
```

```
ptr_int = (int *) calloc ( 10 , sizeof (int) );      /* Aloca 10 inteiros em seqüência */  
ptr_pf = (double *) calloc (10, sizeof (double) ); /* Aloca 10 nros. tipo double */  
free (ptr_int);                                     /* Libera a área de memória alocada */
```

### Fazendo a criação de um vetor de duas formas equivalentes:

```
int tabela[10]; /* Aloca memória: tabela aponta para o início do vetor */  
ou  
int *tabela; tabela = (int *) calloc (10, sizeof(int));
```

## Vetores: Ponteiros e Endereços

### Exemplo: Tabela[10]

```
#include <stdio.h>  
#include <conio.h>  
  
double Tabela[10];  
  
main ()  
{  
    int i;  
  
    for (i=0; i < 10; i++)  
    {  
        printf("Dado %d = ",i);  
        scanf ("%lf",&(Tabela[i]));  
    }  
  
    printf("\nDados Lidos:\n");  
    for (i=0; i < 10; i++)  
        printf("Dado %d = %.2lf\n",i, Tabela[i]);  
  
    getch();  
}
```

### Exemplo: \*Tabela

```
#include <stdio.h>  
#include <conio.h>  
#include <stdlib.h> /* Inclui a biblioteca do "calloc" */  
  
double *Tabela; /* Cria somente o Ponteiro */  
/* Sem alocar memória */  
  
main ()  
{  
    int i;  
  
    Tabela=(double *)calloc(10,sizeof(double));  
  
    for (i=0; i < 10; i++)  
    {  
        printf("Dado %d = ",i);  
        scanf ("%lf",&(Tabela[i]));  
    }  
  
    printf("\nDados Lidos:\n");  
    for (i=0; i < 10; i++)  
        printf("Dado %d = %.2lf\n",i, Tabela[i]);  
  
    getch();  
}
```

## Alocação Dinâmica de Memória

### Exercícios

1. Faça um programa para armazenar em memória um vetor de dados contendo 1500 valores do tipo *int*, usando a função de alocação dinâmica de memória **CALLOC**.
  - 1a) Faça um loop e verifique se o vetor contém realmente os 1500 valores inicializados com zero (conte os 1500 zeros do vetor).
  - 1b) Atribua para cada elemento do vetor o valor do seu índice junto a este vetor. Exibir na tela os 10 últimos elementos do vetor.
2. Faça um programa que pergunte ao usuário quantos valores ele deseja armazenar em um vetor de *doubles*, depois use a função **MALLOC** para reservar (alocar) o espaço de memória de acordo com o especificado pelo usuário. Use este vetor dinâmico como um vetor comum, atribuindo aos 10 primeiros elementos do vetor valores aleatórios (*rand*) entre 0 e 100. Exiba na tela os valores armazenados nos 10 primeiros elementos do vetor.



### INFORMAÇÕES SOBRE A DISCIPLINA

**USP - Universidade de São Paulo - São Carlos, SP**  
**ICMC - Instituto de Ciências Matemáticas e de Computação**  
**SSC - Departamento de Sistemas de Computação**

**Prof. Fernando Santos OSÓRIO**

**Web institucional:** <http://www.icmc.usp.br/ssc/>

**Página pessoal:** <http://www.icmc.usp.br/~fosorio/>

**E-mail:** [fosorio \[at\] icmc. usp. br](mailto:fosorio[at]icmc.usp.br) ou [fosorio \[at\] gmail. com](mailto:fosorio[at]gmail.com)

**PAE Daniel Sales – E-mail:** [dsales \[at\] icmc.usp.br](mailto:dsales[at]icmc.usp.br)

**Monitor Danilo Alvares – E-mail:** [daniilo \[at\] grad.icmc.usp.br](mailto:daniilo[at]grad.icmc.usp.br)

**Disciplina de Introdução a Ciência da Computação**

**Web disciplina:** [Wiki ICMC - Http://wiki.icmc.usp.br](http://wiki.icmc.usp.br)

**> Programa, Material de Aulas, Critérios de Avaliação,**

**> Trabalhos Práticos, Datas das Provas, Notas**