Universidade de São Paulo - São Carlos,SP
Instituto de Ciências Matemáticas e de Computação

USP - ICMC - SSC
SSC 0300 - 2o. Semestre 2013

Disciplina de
Linguagem de Programação e Aplicações
[Eng. Elétrica / Automação]

Prof. Dr. Fernando Santos Osório / PAE: Rafael Klaser (LRM / ICMC)
LRM - Laboratório de Robótica Móvel do ICMC / CROB-SC
Email: fosorio@icmc. usp. br ou fosorio@ gmail. com
Página Pessoal: http://www.icmc.usp.br/~fosorio/

Material on-line:
Wiki ICMC - http://wiki.icmc.usp.br/index.php
Wiki SSC0300 - http://wiki.icmc.usp.br/index.php/SSC-300-2013(fosorio)

Aula 03

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 a Atualizadas:
Consulte REGULARMENTE o material disponível na WIKI
http://wiki.icmc.usp.br/index.php/SSC-300-2013(fosorio)

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*

Memória: Endereços e Ponteiros

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 */
```


Memória: Endereços e Ponteiros

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

Memória: Endereços e Ponteiros

Endereços e Ponteiros:

Declarando Ponteiros:

```
int       *Ponteiro_Int;          /* Ponteiro para um inteiro  */
double  *Ponteiro_Double;         /* Ponteiro para um double  */
FILE    *Arquivo;                 /* Ponteiro para arquivo     */
t_reg_funcionario  *Ptr_Func;     /* Ponteiro para um registro */
                                  /* Criado com typedef/struct */
```

Usando Ponteiros:

& Obtém o endereço de uma variável. Ponteiros são "tipados".

Exemplos: Ponteiro_Int = &Variavel_Int;

Ponteiro_Double = Vetor_Double; /* ou & (Vetor_Double[0]) */

***** Acessa o conteúdo de um endereço qualquer (ler ou escrever no end.)

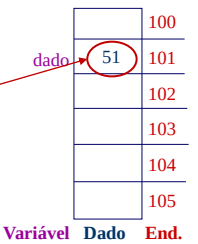
```
*Ponteiro_Int = 10;      /* Escreve o valor 10 no endereço de Ponteiro_Int */
Variavel_Int = *Ponteiro_Int;  /* Lê o valor contido no endereço e atribui
                                  o valor lido para Variavel_Int  */
```


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;
```

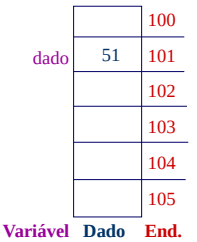
Ponteiros são TIPADOS: Apontam para um determinado tipo de dado

Declaração:	Utilização:	
{ int *ptr_valint;	*ptr_valint = dado_int;	/* escreve */
double *ptr_valreal; ...	ptr_valreal = &dado_real;	/* aponta */

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]       */
}
```


Vetores: Ponteiros e Endereços

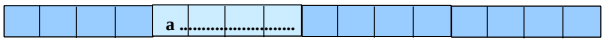
Ponteiros na Linguagem "C" :

- 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;
```

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 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>
```

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	?	?	?

USP – SSC0300 - Semestre 2013-2
Linguagem de Programação e Aplicações
Prof. Fernando Osório / PAE Rafael Klaser

Vetores: Ponteiros e Endereços

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

Tipos de Dados em "C" : Vetores

double *Notas;
*Notas=10.0

- 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;
```

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

- Vetores de caracteres:

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

```
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 reser
                                     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>
```

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);

USP – SSC0300 - Semestre 2013-2
Linguagem de Programação e Aplicações
Prof. Fernando Osório / PAE Rafael Klaser

Vetores: Ponteiros e Endereços

- **Alocação Dinâmica 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*>)

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 <stdlib.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]);

  system("pause");
}
```

Exemplo: *Tabela

```
#include <stdio.h>
#include <stdlib.h>   /* Inclui a biblioteca do "calloc"  */

double *Tabela;       /* Cria somente o Ponteiro */
                      /*  Sem alocar memoria     */
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]);

  system("pause");
}
```

Alocação Dinâmica de Memória

Exercícios

1. **Faça um programa que inicialmente pergunte ao usuário quantos valores ele deseja armazenar em um vetor de *doubles*, depois use a função CALLOC 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 deste vetor valores gerados aleatoriamente (rand), entre 0 e 100. Exiba na tela os valores armazenados nos 10 primeiros elementos do vetor.

   Troque o conteúdo do primeiro elemento do vetor com o conteudo do último elemento do vetor e mostre eles na tela.

Universidade de São Paulo - São Carlos,SP
Instituto de Ciências Matemáticas e de Computação

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/
Página pessoal: http://www.icmc.usp.br/~fosorio/
Página do Grupo de Pesquisa: http://www.lrm.icmc.usp.br/
E-mail: fosorio [at] icmc. usp. br ou fosorio [at] gmail. com

Disciplina de Linguagem de Programação e Aplicações SSC300
WIKI - http://wiki.icmc.usp.br/index.php/SSC-300-2013(fosorio)
> Programa, Material de Aulas, Critérios de Avaliação,
> Trabalhos Práticos, Datas das Provas, Notas