Linguagem de Programação C

PROFESSOR: DIEGO RICARDO KROHL
diego.krohl@ifc.edu.br

Linguagem C – Ponteiros

- O Que é uma variável?
  - É uma área da memória do computador onde é armazenado um valor....

- Exemplo 1:

```
int a = 1;
```

Variável	Posição
a	1000

Linguagem C – Ponteiros

- O Que é um Ponteiro?
  - É uma variável que armazena o endereço na memória do computador onde está outra variável...

- Operadores relacionados a Ponteiros:

  *(asterisco): informa que uma variável irá armazenar o endereço de outra variável; ou:

  informa ao computador que você deseja o valor que está no endereço armazenado;

  &(e comercial): retorna o endereço de uma variável;

Linguagem C – Ponteiros

- Exemplo:

```
int x;
x=10;
int *ponteiro;

ponteiro = &x;

printf ("%i",*ponteiro);

// com asterisco acessa o valor e sem acessa apenas o endereço de memória
```

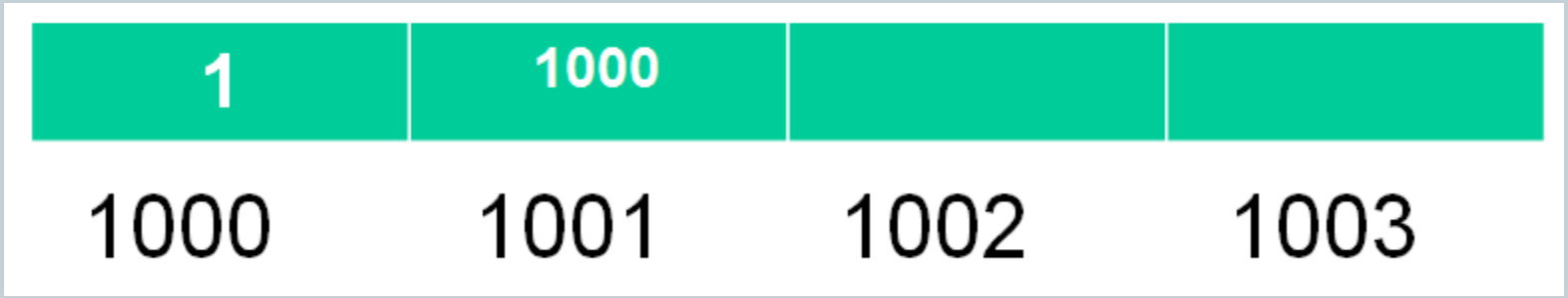
Linguagem C – Ponteiros

- Reforçando:
- operador *
  - Declara-se com *
    - int *x;
  - Acessa-se (alterar, ler) também com *
    - *x = 10; // *atribui o valor 10 ao local apontado pelo ponteiro 'x'*
    - printf("%i", *x); // *imprime o valor armazenado no local apontado por 'x'*
  - Observação: strings e vetores funcionam de forma diferente: um vetor ou string é um ponteiro por definição
- operador &
  - Acessa-se (alterar, ler) o endereço de uma variável (que é um ponteiro)

Linguagem C – Ponteiros

- Exemplo:

```
int a = 1;
int *pt_a;
pt_a = &a;
```



Variável	Posição
a	1000
pt_a	1001

Linguagem C – Ponteiros

- Onde usar:
  - Funções;
  - Alocação Dinâmica
    - Não sei o tamanho que o vetor precisa ter....!
    - Não sei o tamanho que cada string precisa ter...
    - Não sei o tamanho que o vetor / matriz precisa ter...

Linguagem C – Ponteiros

- Exemplo:

```
#include <stdio.h>

main (){
        int num;
        int valor;
        int *p;

        num=55;
        p=&num; /* Pega o endereco de num */
        valor=*p; /* Valor é igualado a num de uma maneira indireta */
        printf ("%i\n",valor);
        printf ("Endereco para onde o ponteiro aponta: %p\n",p);
        printf ("Valor da variavel apontada: %i\n",*p);
        }
```

Linguagem C – Ponteiros

- Exemplo:

```
#include <stdio.h>
        main (){
                int num,*p;
                num=55;
                p=&num; /* Pega o endereco de num */

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

Linguagem C – Ponteiros

- Igualando ponteiros:
  - ```
    int *p1, *p2;
    p1=p2;
    ```
  - Repare que estamos fazendo com que p1 aponte para o mesmo lugar que p2.

- Fazendo com que a variável apontada por p1 tenha o mesmo conteúdo da variável apontada por p2

  ```
  *p1=*p2;
  ```
  ```

Alocação dinâmica de memória

- Durante a execução de um programa é possível alocar uma certa quantidade de memória para conter dados do programa;
- A função malloc (n) aloca dinamicamente n bytes e devolve um ponteiro para o início da memória alocada;
- A função free(p) libera a região de memória apontada por p. O tamanho liberado está implícito, isto é, é igual ao que foi alocado anteriormente por malloc.

Alocação dinâmica de memória

- Os comandos abaixo alocam dinamicamente um inteiro e depois o liberam:

```
#include <stdlib.h>
int *pi;
pi = (int *) malloc (sizeof(int));
...
free(pi);
```

- A função malloc não tem um tipo específico. Assim, (int *) converte seu valor em ponteiro para inteiro. Como não sabemos necessariamente o comprimento de um inteiro (2 ou 4 bytes dependendo do compilador), usamos como parâmetro a função sizeof(int).

Alocação dinâmica de memória

- Exemplo:

```
#include <stdlib.h>
main() {
        int *v, i, n;
        scanf("%i", &n); // le n
        //aloca n elementos para v
        v = (int *) malloc(n*sizeof(int));
        // zera o vetor v com n elementos

        for (i = 0; i < n; i++)
                v[i] = 0;
                ...
        // libera os n elementos de v
        free(v);
}
```

Alocação dinâmica de memória - Exemplo

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

typedef struct {
    int hora;
    int minutos;
    int segundos;
} Horario;

typedef struct {
    int dia;
    int mes;
    int ano;
} Data;

typedef struct {
    Data data;
    Horario horario;
    char descricao[100];
} Compromisso;

```

Alocação dinâmica de memória - Exemplo

```
23  // Função para ler um horário
24  Horario lerHorario() {
25      Horario h;
26      printf("Digite a hora (0-23): ");
27      scanf("%d", &h.hora);
28      printf("Digite os minutos (0-59): ");
29      scanf("%d", &h.minutos);
30      printf("Digite os segundos (0-59): ");
31      scanf("%d", &h.segundos);
32      return h;
33  }
34
35  // Função para ler uma data
36  Data lerData() {
37      Data d;
38      printf("Digite o dia: ");
39      scanf("%d", &d.dia);
40      printf("Digite o mês: ");
41      scanf("%d", &d.mes);
42      printf("Digite o ano: ");
43      scanf("%d", &d.ano);
44      return d;
45  }
```

Alocação dinâmica de memória - Exemplo

```
47  // Função para ler um compromisso
48  Compromisso lerCompromisso() {
49      Compromisso c;
50      c.data = lerData();
51      c.horario = lerHorario();
52      printf("Digite a descrição do compromisso: ");
53      scanf(" %99[^\n]", c.descricao);
54      return c;
55  }
56
57  // Função para exibir um compromisso
58  void exibirCompromisso(Compromisso c) {
59      printf("Compromisso em %02d/%02d/%04d às %02d:%02d:%02d\n",
60             c.data.dia, c.data.mes, c.data.ano,
61             c.horario.hora, c.horario.minutos, c.horario.segundos);
62      printf("Descrição: %s\n", c.descricao);
63  }
```

Alocação dinâmica de memória - Exemplo

```c
main() {
    int n, i;
    printf("Quantos compromissos deseja criar? ");
    scanf("%d", &n);

    // Alocação dinâmica de memória para os compromissos
    Compromisso *compromissos = (Compromisso *)malloc(n * sizeof(Compromisso));
    if (compromissos == NULL) {
        printf("Erro de alocação de memória!\n");
        return 1;
    }

    for (i = 0; i < n; i++) {
        printf("\nCompromisso %d:\n", i + 1);
        compromissos[i] = lerCompromisso();
    }

    printf("\nCompromissos criados:\n");
    for (i = 0; i < n; i++) {
        printf("\nCompromisso %d:\n", i + 1);
        exibirCompromisso(compromissos[i]);
    }

    // Liberação da memória alocada
    free(compromissos);
}
```

Alocação dinâmica de memória - Exemplo

- E se eu não quiser definir a quantidade de compromissos (tamanho)?
  - Usar lista encadeada (próximas aulas)...

Alocação dinâmica de memória - Exercícios

- Elabore um programa que declare um vetor com 7 elementos, preencha-o com alguns valores e mostre na tela, para cada elemento do vetor: seu valor e seu endereço de memória utilizando ponteiros.

Alocação dinâmica de memória - Exercícios

```c
#include <stdio.h>

main() {
    // Declaração do vetor com 7 elementos
    int vetor[7];

    // Preenchimento do vetor com alguns valores
    for (int i = 0; i < 7; i++) {
        vetor[i] = i * 10; // Exemplo de preenchimento: múltiplos de 10
    }

    // Mostrando valor e endereço de memória de cada elemento
    for (int i = 0; i < 7; i++) {
        printf("Valor do elemento %d: %d\n", i, vetor[i]);
        printf("Endereco de memoria do elemento %d: %p\n", i, &vetor[i]);
        //%p -> imprimir um endereço
    }
}
```

Alocação dinâmica de memória - Exercícios

- Seguindo o exercício anterior, através do ponteiro multiplique todos os valores do vetor por 2, e reimprima seu novo valor e endereço.

Alocação dinâmica de memória - Exercícios

```
19      // Multiplicando todos os valores do vetor por 2 usando ponteiros
20      for (int i = 0; i < 7; i++) {
21          int *ptr = &vetor[i]; // Ponteiro para o elemento do vetor
22          *ptr = *ptr * 2;      // Multiplicando o valor apontado por 2
23      }
24
25      // Mostrando os novos valores e endereços de memória de cada elemento
26      printf("\nNovos valores e endereços de memória após a multiplicação:\n");
27      for (int i = 0; i < 7; i++) {
28          printf("Novo valor do elemento %d: %d\n", i, vetor[i]);
29          printf("Endereço de memória do elemento %d: %p\n", i, &vetor[i]);
30      }
31  }
```