Pontifícia Universidade Católica
do Rio de Janeiro

INF 1005 – Programação I

Aula 08 – Ponteiros

Edirlei Soares de Lima
<elima@inf.puc-rio.br>

visionlab
visualization laboratory

Ponteiros

- **Ponteiro é tipo especiais de dado que armazenam um endereços de memória.**

- Uma variável do tipo ponteiro deve ser declarada da seguinte forma:

```
tipo *nome_variavel;
```

- A variável ponteiro armazena um endereço de memória de uma outra variável do tipo especificado.

```
int *ponteiro_a; float *ponteiro_b;
```

Operadores de Ponteiros

- Existem dois **operadores** relacionados a ponteiros:
  - O operador **&** retorna o endereço de memória de uma variável:

```
int *p;
int a = 40;
p = &a;
```

  - O operador * retorna o conteúdo do endereço indicado pelo ponteiro:

```
printf("%d", *p);
```

Operadores de Ponteiros

```c
#include <stdio.h>

int main(void)
{
  int a;
  int *p; /* declaração */
  p = &a; /* inicialização */
  a = 0;
  *p = 2;
  printf("%d", a);
  return 0;
}
```

SAIDA:

2

Operadores de Ponteiros

```
#include <stdio.h>

int main(void)
{
  int a;
  int *p = &a; /*declaração e inicialização*/
  *p = 10;
  printf("%d", a);
  return 0;
}
```

SAIDA:

10

Operadores de Ponteiros

```
#include <stdio.h>

int main(void)
{
  int num, q = 1;
  int *p;
  num = 100;
  p = &num;
  q = *p;
  printf("%d", q);
  return 0;
}
```

SAIDA:

100

Cuidados com Ponteiros

- Não se pode atribuir um valor ao endereço apontado pelo ponteiro, sem antes ter certeza de que o endereço é válido:

```
int a;
int *c;

*c = 20; /* armazenará 20 em qual endereço? */
```

- O correto seria:

```
int a;
int *c;
c = &a;
*c = 13;
```

Cuidados com Ponteiros

- Como o operador * de ponteiros é igual ao operador * utilizado na multiplicação, deve-se ter cuidado no uso desses operadores:

```
#include <stdio.h>
int main()
{
  int b, a;
  int *c;
  b = 10;
  c = &a;
  *c = 11;
  a = b * c;
  printf("%d", a);
  return 0;
}
```

Ocorre um erro de compilação, pois o * é interpretado como operador de ponteiro sobre c

Cuidados com Ponteiros

- Para corrigir é necessário isolar o operador de ponteiro na expressão:

```
#include <stdio.h>
int main()
{
  int b, a;
  int *c;
  b = 10;
  c = &a;
  *c = 11;
  a = b * (*c);
  printf("%d", a);
  return 0;
}
```

Operações com Ponteiros

```
#include <stdio.h>
int main()
{
  float *a, *b, c, d;
  a = &d;
  b = &c;
  scanf("%f %f", &c, &d);
  if (b < a)
    printf("Endereco apontado por b é menor: %p e %p\n", b, a);
  else if (a < b)
    printf("Endereco apontado por a é menor: %p e %p\n", a, b);
  else if (a == b)
    printf("Mesmo endereco\n"); /* sempre diferentes! */
  if (*a == *b)
    printf("Mesmo conteudo: %f\n", *a);
  return 0;
}
```

Passagem de Parâmetros

- Os parâmetros de uma função são o mecanismo utilizado para passar a informação de um trecho de código para o interior da função.

- **Ha dois tipos de passagem de parâmetros:**
  - Passagem por valor.
  - Passagem por referência.

Passagem de Parâmetros por Valor

```
#include <stdio.h>
void troca(int a, int b)
{
  int tmp = b;
  b = a;
  a = tmp;
}
int main (void)
{
  int a = 10, b = 20;
  troca(a, b);
  printf("A=%d B=%d", a, b);
}
```

SAIDA:

A = 10 B = 20

Passagem de Parâmetros por Referência

- A linguagem C permite a **passagem de ponteiros para funções**.
- Isso permite que as funções possam **alterar o conteúdo das variáveis** indiretamente pelo seu endereço de memória.
- Se uma função g chama uma função f:
  - f não pode alterar diretamente valores de variáveis de g, porém
  - se g passar para f os valores dos endereços de memória onde as variáveis de g estão armazenadas, f pode alterar, indiretamente, os valores das variáveis de g.

Passagem de Parâmetros por Referência

```
#include <stdio.h>
void troca(int *a, int *b)
{
  int tmp = *b;
  *b = *a;
  *a = tmp;
}
int main (void)
{
  int a = 10, b = 20;
  troca(&a, &b);
  printf("A=%d B=%d", a, b);
}
```

SAIDA:

A = 20 B = 10

Passagem de Parâmetros por Referência

- Vantagens da utilização de parâmetros por referência:
  - **Mais eficiência:** as funções recebem os endereços para as variáveis já inicializadas e o tamanho do endereço é sempre o mesmo, assim não há problema envolvendo cópias e inicialização.
  - **Mais liberdade:** possibilita a criação de funções que podem retornar mais do que um valor.

Exercícios

Lista de Exercícios 06 - Ponteiros

http://www.inf.puc-rio.br/~elima/prog1/