MC-102 — Aula 13
Funções e Procedimentos II

Instituto de Computação – Unicamp

Segundo Semestre de 2011

Roteiro

Ponteiro

- Ponteiros são tipos especiais de dados que armazenam endereços de memória.
- E para que serve? Serve para que possamos acessar uma porção definida da memória.
- Para declarar um variável ponteiro devemos fazer:
  - tipo *nome-variável;
- Exemplo
  - int *p;
  - Onde p é o nome do ponteiro e "int *" diz ao compilador que p guardará um endereço de memória onde será armazenado um inteiro.

Ponteiro

- Mas, e que endereços esses ponteiros possuem?
- Até agora nenhum, ou melhor, algum lixo que esteja na memória ocupada por p. E como fazemos para "zerar" um ponteiro?
  - p = NULL;

Ponteiro

- NULL está definida na biblioteca "stdlib.h"e tem o valor 0. Se não fizermos isso, p poderá conter um endereço de memória que pode já ter sido liberada pelo programa, ou que pode não pertencer ao programa, o que geraria um erro quando da execução.
- Como damos um endereço a ele? Antes de mais nada, geralmente não queremos dar um endereço qualquer, mas sim o endereço de alguma variável. E como passamos o endereço de uma variável a um ponteiro?

Operação com ponteiros

- Existem dois operadores relacionados a ponteiros:
- O operador & retorna o endereço de memória de uma variável:
  - int *mema;
  - int a=90;
  - mema = &a;
- O operador * retorna o valor do conteúdo no endereço que segue:
  - printf("%d", *mema);

Exemplos Ponteiro

```
#include <stdio.h>
int main(){
int num, q;
int *p;
num=100;
p = &num;
q = *p;
printf("%d",q);
}
```

Exemplos Ponteiro

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

CUIDADO

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

```
int a,b;
int *c;
b=10;
*c=13; //Vai colocar em qual endereço?
```

- O correto seria por exemplo:

```
int a,b;
int *c;
b=10;
c = &a;
*c=13;
```

CUIDADO

- Infelizmente o operador * de ponteiros é igual a multiplicação portanto preste atenção em como utilizá-lo em algumas expressões.

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

- O que ocorre se tentarmos compilar este código?.

CUIDADO

- Ocorre um erro de compilação pois o * é interpretado como operador de ponteiro sobre c.
- O correto seria algo como:

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

CUIDADO

- O endereço que um ponteiro armazena é sempre de um tipo específico

```
#include <stdio.h>
int main(){
double b,a;
int *c;
b=10.89;
c=&b; //ops
a=*c;
printf("\n%f\n",a);
}
```

- Além do compilador alertar que a atribuição pode causar problemas é impresso um valor totalmente diferente de 10.89.

EXEMPLO DE COMPARAÇÃO ENTRE PONTEIROS

- Você pode fazer comparações entre ponteiros:

```
#include <stdio.h>
int main(){
double *a,*b, c,d;
b=&c;
a=&d;
if(b<a)
printf("\nO endereco apontado por b e menor:%p e %p",b,a);
else if(a<b)
printf("\nO endereco apontado por a e menor:%p e %p",a,b);
else if(a == b)
printf("Mesmo endereco");
if(*a == *b)
printf("Mesmo conteudo");
}
```

Funções

Voltando para as Funções

- Passagem de parametros

Passagem de argumentos por valor

- Quando passamos argumentos a uma função, os valores fornecidos são copiados para os parâmetros formais da função. Este processo é idêntico a uma atribuição.
- Desta forma, alterações nos parâmetros dentro da função não alteram os valores que foram passados:

```
void nao_troca(int x, int y) {
  int aux;
  aux = x;
  x = y;
  y = aux;
}
```

Veja o exemplo em `nao_troca.c`.

Passagem de argumentos por referência

- Existe uma forma de alterarmos a variável passada como argumento, ao invés de usarmos apenas o seu valor.
- O artifício corresponde a passarmos como argumento o endereço da variável, e não o seu valor.
- Para indicarmos que será passado o endereço do argumento, usamos o mesmo tipo que usamos para declarar um variável que guarda um endereço:

```
tipo nome (tipo *parâmetro1, tipo *parâmetro2, ...,
           tipo *parâmetroN) {
    comandos;
}
```

Passagem de argumentos por referência

- Um endereço de variável passado com parâmetro não é muito útil. Para acessarmos o valor de uma variável apontada por um endereço, usamos o operador *:
- Ao precedermos uma variável que contém um endereço com este operador, obtemos o equivalente a variável armazenada no endereço em questão:

```
void troca(int *end_x, int *end_y) {
  int aux;
  aux = *end_x;
  *end_x = *end_y;
  *end_y = aux;
}
```

Veja o exemplo em `troca.c`.

Passagem de argumentos por referência

- Uma outra forma de conseguirmos alterar valores de variáveis externas a funções é usando variáveis globais.
- Nesta abordagem usamos variáveis globais no lugar de parâmetros e de valores de retorno.
- Porém, ao usar esta técnica estamos negando uma das principais vantagens de se usar funções, reaproveitamento de código.

Veja um exemplo em `vetor_global.c`.

Exercício

- Implemente um programa que calcule o quadrado de um número. Utilize o conceito de funções. Como ficaria o programa se for utilizado o conceito de ponteiros? Implemente um programa que calcule a e-nesima potência de x.