MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2013

Roteiro

1. Ponteiros
2. Passagem de parâmetros por valor e por referência
3. Aritmética de ponteiros
4. Ponteiros e vetores
5. Ponteiros de ponteiros
6. Exercícios

Ponteiro

- Ponteiros são tipos especiais de dados que armazenam endereços de memória.
- Uma variável do tipo ponteiro deve ser declarada da seguinte forma:

```
tipo *nome_variável;
```

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

Exemplos:

```
int *mema;  float *memb;
```

mema armazena endereço de memória de variáveis do tipo `int`.
memb armazena endereço de memória de variáveis do tipo `float`.

Operadores de ponteiro

- 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 conteúdo do endereço indicado pelo ponteiro:

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

Operadores de ponteiro

```
#include <stdio.h>

int main() {
  int b;
  int *c;

  b = 10;
  c = &b;
  *c = 11;
  printf("%d\n", b);
  return 0;
}
```

O que será impresso?

11

Operadores de ponteiro

```
#include <stdio.h>

int main() {
  int num, q = 1;
  int *p;

  num = 100;
  p = &num;
  q = *p;

  printf("%d\n", q);
  return 0;
}
```

O que será impresso?
100

Cuidado!

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

```
int a, b;
int *c;

b = 10;
*c = 13;  /* armazenará 13 em qual endereço? */
```

- O correto seria, por exemplo:

```
int a, b;
int *c;

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

Cuidado!

- 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\n", a);
  return 0;
}
```

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

Cuidado!

- 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("%d\n", a);
  return 0;
}
```

Cuidado!

- O ponteiro armazena o endereço de um tipo específico.

```
#include <stdio.h>

int main() {
  double b, a;
  int *c;

  b = 10.89;
  c = &b;  /* tipos diferentes! */
  a = *c;
  printf("%lf\n",a);
  return 0;
}
```

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

Operações com ponteiros

Pode-se fazer comparações entre ponteiros ou o conteúdo apontado por estes:

```
#include <stdio.h>

int main() {
  double *a, *b, c, d;
  a = &d;
  b = &c;
  scanf("%d %d", &c, &d);

  if (b < a)
    printf("Endereco apontado por b eh menor: %p e %p\n", b, a);
  else
    if (a < b)
      printf("Endereco apontado por a eh menor: %p e %p\n", a, b);
    else if (a == b)
      printf("Mesmo endereco\n"); /* sempre diferentes! */
  if (*a == *b)
    printf("Mesmo conteudo: %lf\n", *a);
  return 0;
}
```

Notem que, para imprimir um ponteiro, usamos %p.

Operações com ponteiros

- Quando um ponteiro não está associado a nenhum endereço válido é comum atribuir o valor `NULL` para este (definido na biblioteca `stdlib.h`).
- Isto é usado em comparações com ponteiros para saber se um determinado ponteiro possui valor válido ou não.

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

int main() {
  double *a = NULL, *b, c = 5;
  a = &c;

  if (a != NULL) {
    b = a;
    printf("Numero : %lf\n", *b);
  }

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

Passagem de parâmetros por valor

- Quando passamos parâmetros a uma função, os valores fornecidos são copiados para as variáveis parâmetros da função.
- Este processo é idêntico a uma atribuição e é chamado de passagem por valor.
- Desta forma, alterações nos parâmetros dentro da função não alteram os valores que foram passados.

```
#include <stdio.h>
void nao_troca(int a, int b) {
  int aux;
  aux = a;
  a = b;
  b = aux;
}

int main() {
  int x = 4, y = 5;
  nao_troca(x, y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Passagem de parâmetros por referência

- Em C, só existe passagem de parâmetros por valor.
- Em algumas linguagens, há construções para se passar *parâmetros por referência*.
  - Neste último caso, alterações de um parâmetro passado por referência também ocorrem onde foi feita a chamada da função.
  - No exemplo anterior, se `x` e `y` fossem passados por referência, seus conteúdos seriam trocados.

Passagem de parâmetros por referência

- Podemos obter algo semelhante em C utilizando ponteiros.
- O artifício corresponde em passar como parâmetro para uma função o *endereço* da variável e não o seu valor.
- Desta forma, podemos alterar o conteúdo da variável como se fizéssemos passagem por referência.

```
#include <stdio.h>
void troca(int *a, int *b) {
  int aux;
  aux = *a;
  *a = *b;
  *b = aux;
}

int main() {
  int x = 4, y = 5;
  troca(&x, &y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Passagem de parâmetros por referência

- O uso de ponteiros para passar parâmetros que devem ser alterados dentro de uma função é útil em certas situações como:
  - Funções que precisam retornar mais do que um valor.
- Suponha que queremos criar uma função que recebe um vetor como parâmetro e precisa retornar o maior e o menor elemento do vetor.
  - Uma função só retorna um único valor!
  - Podemos passar ponteiros para variáveis que armazenarão o maior e menor elemento.

Passagem de parâmetros por referência

```
#include <stdio.h>
void min_and_max(int vet[], int tam, int *min, int *max);

int main() {
  int v[] = {10, 80, 5, -10, 45, -20, 100, 200, 10};
  int min, max;
  min_and_max(v, 9, &min, &max);
  printf("Menor valor: %d\n", min);
  printf("Maior valor: %d\n", max);
  return 0;
}

void min_and_max(int vet[], int tam, int *min, int *max) {
  int i;
  *min = vet[0];
  *max = vet[0];
  for (i = 1; i < tam; i++) {
    if (vet[i] < *min)
      *min = vet[i];
    if (vet[i] > *max)
      *max = vet[i];
  }
}
```

Aritmética de ponteiros

- Os operadores + e - podem ser utilizados com ponteiros.
- Seja `p` um ponteiro para um inteiro com um valor atual de 3000.
- A expressão:

  `p++;`

  faz com que o conteúdo de `p` seja alterado para 3004 (e não 3001).
- A cada incremento de `p`, ele apontará para o próximo endereço de um tipo inteiro, cujo tamanho é de 4 bytes para o computador neste exemplo.

Aritmética de ponteiros

- O mesmo é válido para decrementos.
- Por exemplo:

  `p--;`

  fará com que `p` assuma o valor 2996 (considerando que o endereço anterior era 3000 e o tamanho de um inteiro é de 4 bytes).

Ponteiros e vetores

- Quando declaramos uma variável do tipo vetor, aloca-se uma quantidade de memória contígua cujo tamanho é especificado na declaração (e também depende do tipo do vetor).
  - `int v[5];` - serão alocados 5*4 bytes de memória associada com `v`.
- Uma variável vetor, assim como um ponteiro, armazena um endereço de memória: o endereço de início do vetor.
  - `int v[5];` - variável `v` contém o endereço de memória do início do vetor.
- Por este motivo, quando passamos um vetor como parâmetro para uma função, seu conteúdo pode ser alterado dentro da função (pois estamos passando, na realidade, o endereço de início do espaço alocado para o vetor).

Ponteiros e vetores

```
#include <stdio.h>

void zeraVetor(int vet[], int tam) {
  int i;
  for (i = 0;  i < tam; i++)
    vet[i] = 0;
}

int main() {
  int vetor[] = {1, 2, 3, 4, 5};
  int i;

  zeraVetor(vetor, 5);
  for (i = 0; i < 5; i++)
    printf("%d ", vetor[i]);
  printf("\n");
  return 0;
}
```

Ponteiros e vetores

- De fato, como uma variável vetor possui um endereço, podemos atribui-la a uma variável ponteiro:

```
int a[] = {1, 2, 3, 4, 5};
int *p;
p = a;
```

- E podemos então usar `p` como se fosse um vetor:

```
for (i = 0; i < 5; i++)
  p[i] = i*i;
```

Ponteiros e vetores: diferenças!

- Uma variável vetor, diferentemente de um ponteiro, possui um endereço fixo.
- Isto significa que você não pode tentar atribuir um endereço a uma variável do tipo vetor.

```
#include <stdio.h>

int main() {
  int a[] = {1, 2, 3, 4, 5};
  int b[5], i;

  b = a;
  for (i = 0; i < 5; i++)
    printf("%d ", b[i]);
  printf("\n");
  return 0;
}
```

Ocorre erro de compilação!

Ponteiros e vetores: diferenças!

- Entretanto, se b for declarado como ponteiro, não há problemas:

```
#include <stdio.h>

int main() {
  int a[] = {1, 2, 3, 4, 5};
  int *b, i;

  b = a;
  for (i = 0; i < 5; i++)
    printf("%d ", b[i]);
  printf("\n");
  return 0;
}
```

Ponteiros e vetores: diferenças!

- Aritmética de ponteiros pode ser utilizada com vetores.

  Exemplo:

```
char str[80];
char *p;
p = str;
```

  `p` foi definido como o endereço do primeiro elemento do vetor (string) `str`.

  Para fazer acesso ao quinto elemento de `str`, pode-se escrever:

```
str[4];
```

  ou

```
*(p + 4);
```

Ponteiros e vetores: diferenças!

O resultado impresso por este programa...

```
#include <stdio.h>

int main() {
  int a[] = {1, 2, 3, 4, 5};
  int *b, i;

  b = a;
  for (i = 0; i < 5; i++)
    printf("%d ", a[i]);
  printf("\n");

  return 0;
}
```

Ponteiros e vetores: diferenças!

... é igual a este...

```
#include <stdio.h>

int main() {
  int a[] = {1, 2, 3, 4, 5};
  int *b, i;

  b = a;
  for (i = 0; i < 5; i++)
    printf("%d ", *(a + i));
  printf("\n");

  return 0;
}
```

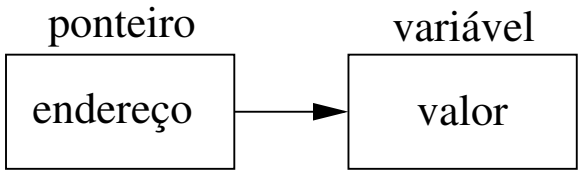
Ponteiros e vetores: diferenças!

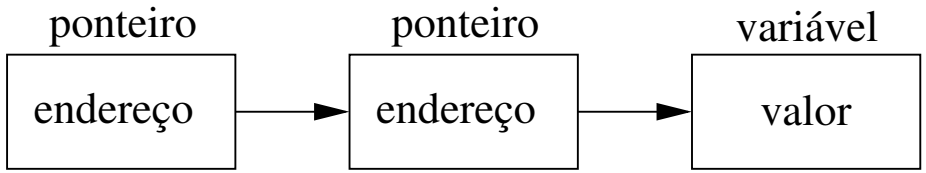
... e a este.

```
#include <stdio.h>

int main() {
  int a[] = {1, 2, 3, 4, 5};
  int *b, i;

  b = a;
  for (i = 0; i < 5; i++)
    printf("%d ", *(b + i));
  printf("\n");

  return 0;
}
```

Ponteiros de ponteiros

- O ponteiro de um ponteiro é uma forma de endereçamento múltiplo.
- Na figura à esquerda, o valor do ponteiro é o endereço da variável que contém o valor desejado.
- Na figura à direita, o primeiro ponteiro contém o endereço de um segundo ponteiro, que aponta para a variável que tem o valor desejado.





Ponteiros de ponteiros

- Exemplo: o que será impresso com a execução do código abaixo?

```
#include <stdio.h>

int main() {
  int x, *p1, **p2;

  x = 100;
  p1 = &x;
  p2 = &p1;
  printf("%d\n", **p2);
  return 0;
}
```

O programa imprimirá o valor de `x`, ou seja, 100.

Exercício

O que será impresso?

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

int main() {
  int a = 3, b = 2, *p = NULL, *q = NULL;

  p = &a;
  q = p;
  *q = *q + 1;
  q = &b;
  b = b + 1;

  printf("%d\n", *q);
  printf("%d\n", *p);
  return 0;
}
```

3

4

Exercício

Escreva a função `strcpy(s, t)` que recebe como parâmetro duas strings e copia a string `t` na string `s`. Observação: assuma que a string `s` possui espaço suficiente para comportar a string `t`.

```
/* versao com vetores */
void strcpy(char s[], char t[]) {
  int i = 0;

  while ((s[i] = t[i]) != '\0')
    i++;
}

/* versao 1 com ponteiros */
void strcpy(char *s, char *t) {

  while ((*s = *t) != '\0') {
    s++;
    t++;
  }
}
```

Exercício

```
/* versao 2 com ponteiros */
void strcpy(char *s, char *t) {

  while ((*s++ = *t++) != '\0');
}

/* versao 3 com ponteiros */
void strcpy(char *s, char *t) {

  while (*s++ = *t++);
}
```

Exercício

Escreva a função `strcmp(s, t)` que recebe como parâmetro duas strings e compara `s` e `t`, retornando um valor negativo, zero ou positivo se `s` é lexicograficamente menor, igual ou maior que `t`, respectivamente.

```
/* versao com vetores */
int strcmp(char s[], char t[]) {
  int i = 0;

  while (s[i] == t[i])
    if (s[i++] == '\0')
      return 0;
  return (s[i] - t[i]);
}

/* versao com ponteiros */
int strcmp(char *s, char *t) {
  while (*s == *t) {
    if (*s == '\0')
      return 0;
    s++; t++;
  }
  return (*s - *t);
}
```

Exercício

Escreva a função `strcat(s, t)` que recebe como parâmetro duas strings e concatena `t` em `s`.

```
/* versao com vetores */
void strcat(char s[], char t[]) {
  int i = 0, j = 0;

  while (s[i] != '\0')
    i++;
  while (t[j] != '\0') {
    s[i] = t[j];
    i++;
    j++;
  }
  s[i] = '\0';
}
```

Exercício

```
/* versao com ponteiros */
void strcat(char *s, char *t) {

  while (*s != '\0')
    s++;
  while (*t != '\0') {
    *s = *t;
    s++;
    t++;
  }
  *s = '\0';
}
```

Exercício

Escreva a função `strlen(s)` que recebe como parâmetro a string `s` e retorna seu comprimento.

```
/* versao com vetores */
int strlen(char s[]) {
  int i = 0;

  while (s[i] != '\0')
    i++;
  return i;
}

/* versao com ponteiros */
int strlen(char *s) {
  int i = 0;

  while (*s != '\0') {
    i++;
    s++;
  }
  return i;
}
```