Introdução à Ciência da Computação

Ponteiros em C

Prof. Ricardo J. G. B. Campello

Sumário

- Introdução
- Definição de Ponteiros
- Declaração de Ponteiros em C
- Manipulação de Ponteiros em C
  - Operações
  - Ponteiros e Arranjos
  - Passagem de Parâmetros por Referência

Introdução

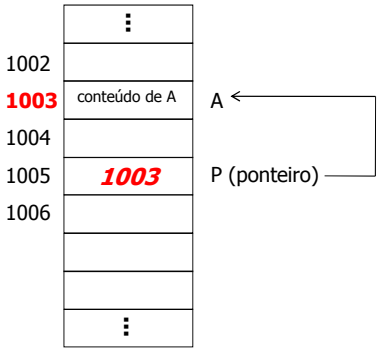
- Por quê ponteiros são importantes?
  - Permitem passagem de parâmetros para funções **por referência**
  - Permitem **alocar e liberar memória** dinamicamente (em tempo de execução)
  - Possibilitam a implementação eficiente de certas **estruturas de dados**

Definição

- Um **ponteiro** (ou **apontador**) é uma variável que armazena um endereço de memória
  - Normalmente, esse endereço é a posição de outra variável na memória
  - Dizemos portanto que um ponteiro "aponta" para uma variável
  - O tipo do ponteiro é normalmente associado ao tipo da variável apontada

Definição

Exemplo:



Declaração em C

- `tipo *var1, *var2, ...;`
  - O símbolo * indica que as variáveis *var1, var2, etc.* são ponteiros para variáveis do tipo *tipo*
- Alternativamente pode-se declarar como:
  - `tipo*  var1, var2, ...;`
- Exemplos:
  - **float** *a, *t23, *lista;
  - **int*** b, s2;

Manipulação em C

- Operadores:
  - **Operador de endereço**: **&**
    - Ponteiros só podem receber endereços de memória
    - Para isso, pode-se utilizar o operador &
  - **Operador de conteúdo**: ***
    - Recupera o conteúdo da variável apontada por um ponteiro
  - **Ponteiro nulo**: **NULL**
    - Um ponteiro que recebe o valor NULL aponta para nada

Manipulação em C

- Atribuição (P1 e P2 ponteiros – A, B e C variáveis):
  - **Endereço de Variável para Ponteiro**
    - Exemplos: P1 = &A; P2 = &B;
  - **Conteúdo de Endereço Apontado para Variável**
    - Exemplo: C = *P2;
  - **Ponteiro para Ponteiro (ou para nulo)**
    - Exemplos: P1 = P2; P2 = **NULL**;
  - **Conteúdo para Endereço Apontado**
    - Exemplos: *P1 = A; *P1 = 34;

Manipulação em C

Exemplo:

```
3.123
3.123
2.456
2.456
1.789
2.456
2.456
1.789
```

```c
#include <stdio.h>
void main(void){
   float A, B, *P1, *P2;
   A = 3.123;
   printf("%f\n", A);
   P1 = &A;
   printf("%f\n",*P1);
   *P1 = 2.456;
   printf("%f\n",A);
   P2 = P1;
   B = *P2;
   printf("%f\n",B);
   B = 1.789;
   P2 = &B;
   printf("%f\n%f\n%f\n%f",B, A, *P1, *P2);
}
```

Manipulação em C

- Comparação:
  - **Como para qualquer variável**
  - Exemplos:
    - **if** (P1 == P2) /* Verifica se P1 e P2 apontam para o mesmo endereço */
    - **if** (P1 == **NULL**) /* Verifica se P1 aponta para **NULL** */

Ponteiros para Estruturas

- Podemos declarar ponteiros para estruturas
- Por exemplo:

```c
struct estrutura {
  float   campo1;
  char    campo2; }
struct estrutura  est, *pt;
```

- Na verdade, é possível declarar ponteiros para tipos definidos pelo usuário em geral

Ponteiros para Estruturas

- No exemplo anterior, poderíamos definir o tipo:

```c
typedef struct {
                float campo1;
                char  campo2; } minha_estrutura;
minha_estrutura  est, *pt;
```

- Em qualquer caso, o acesso aos campos de uma estrutura através de um ponteiro demanda o **operador seta** (–>)

Ponteiros para Estruturas

- No exemplo anterior, poderíamos utilizar o operador seta para acessar os campos da var. composta `est` através de um ponteiro `pt`

```
pt = &est;
printf("O valor do campo1 é %f", pt->campo1);
printf("O valor do campo2 é %c", pt->campo2);
```

- Notas:
  - Operador `->` acessa conteúdo, não endereço !
  - `pt->campo1` é equivalente a `est.campo1`

Ponteiros e Arranjos

- Há uma relação estreita entre ponteiros e arranjos (vetores e matrizes) em linguagem C
- Por exemplo, considere as seguintes declarações:
  - **char** str[10], *p1;
    - vetor de 10 chars (string) e ponteiro para chars
  - As seguintes atribuições são equivalentes:
    - p1 = str **ou** p1 = &str[0]
    - o nome do vetor referencia o endereço do seu 1º elemento
    - o mesmo vale para matrizes, armazenadas linearmente em memória
      - como se fosse um vetor, linha por linha

Ponteiros e Arranjos

- Há uma relação estreita entre ponteiros e arranjos (vetores e matrizes) em linguagem C
- Por exemplo, considere as seguintes declarações:
  - **char** str[] = {'b', 'l', 'a', 'b', 'l', 'a', 'b', 'l', 'a', '\0'}, *p1;
    - vetor de 10 chars (string) e ponteiro para chars
  - As seguintes expressões são equivalentes:
    - str[5] **ou** *(str + 5)
    - ambas representam o conteúdo do sexto elemento da string (“a”)
      - lembrando que arranjos em C são indexados a partir de zero

Ponteiros e Arranjos

- **Nota 1:** embora o nome do arranjo referencie seu 1º elemento, não podemos atribuir ou modificar o valor deste como se fosse um ponteiro qualquer
  - ou perderíamos a referência para o início do arranjo...
- Por exemplo, o seguinte trecho de código <u>não</u> compila:

```
char str[]={'b', 'l', 'a', 'b', 'l', 'a', 'b', 'l', 'a', '\0'};
while (*str != '\0'){ printf("%c", *str); str++; }
```

- Para corrigir:

```
char* p1 = str;
while (*p1 != '\0'){ printf("%c", *p1); p1++; }
```

Ponteiros e Arranjos

- **Nota 2:** a relação entre arranjos e ponteiros é tão próxima que um ponteiro para um arranjo pode ser indexado como se o próprio arranjo fosse (sem *)
- Por ex., o código anterior poderia ser rescrito como:

```
char  str[10] = "blablabla", *p1;
int i;
p1 = str;
for (i=0; i<9; i++) printf("%c", p1[i]);
```


Relembrando Funções...

- Escopo:
  - As variáveis declaradas dentro de uma função possuem **escopo local**, ou seja, provisório e interno à função
  - Variáveis **globais** são declaradas no início de qualquer módulo (arquivo) de programa, fora de qualquer função (incluindo `main`)
- Passagem de Parâmetros:
  - Em geral, **por valor** (uma **cópia** do parâmetro é feita)
    - Em C ANSI, isso inclui *structs*
  - Para forçar passagem por **referência**, é preciso utilizar ponteiros
    - É o que ocorre necessariamente com arranjos (vetores/matrizes)

Funções e Ponteiros

- Passagem **por Referência**
  - Alterações feitas nos parâmetros recebidos são refletidas nos valores passados como argumentos
  - O **endereço** do argumento é passado no momento da chamada da função
    - endereço do argumento = ponteiro

Por que Passagem por Referência?

- Em muitas situações em programação, a passagem de parâmetros **por valor** não é apropriada
- Por exemplo, a função abaixo é inócua:

```
void troca(int x, int y){
  int temp;
  temp = x;
  x = y;
  y = temp; }
```

pois não altera as variáveis passadas como parâmetros, altera apenas as cópias dos seus valores feitas em x e y

- tais cópias desaparecem ao final da execução da função

Passagem de Ponteiros

- Para forçar uma passagem por **referência**, é preciso utilizar ponteiros
- Nesse caso, a função do exemplo anterior fica:

```
void troca(int *x, int *y){
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
```

e troca os valores contidos nos **endereços** passados como parâmetro

Passagem de Ponteiros

- Veja como fica uma chamada à função anterior:

```
void main(void){
  int temp, a = 10, b = 20;
  printf("%d %d\n", a, b);
  troca(&a, &b);
  printf("%d %d", a, b);
}
```

- Isso explica a sintaxe de scanf (ou seja, por que devemos incluir o operador **&** antecedendo as variáveis a receberem valores) ???
  - também explica por que isso não se faz necessário no caso de strings...?

Passagem de Ponteiros

- A passagem de **arranjos** para funções é uma exceção à convenção de passagem por valor da linguagem C
  - Quando uma função possui um arranjo (vetor/matriz) como parâmetro, apenas uma cópia do **endereço** do arranjo é passada na chamada, não uma cópia do arranjo todo
  - Como já sabemos, o endereço do arranjo é um ponteiro para o seu 1º elemento e é referenciado pelo seu próprio nome
- Exemplo:
  - Função para imprimir string sem usar `fprintf("%s", var)`

Passagem de Ponteiros

- Exemplo:

```
void imprime_string(char *st) {
  while (*st != '\0'){ printf("%c", *st); st++; }
```

- Formas alternativas **equivalentes**:
  - `void imprime_string(char* st) { ... }`
  - `void imprime_string(char st[]) { ... }`
- Exemplos de chamada:
  - `imprime_string("alo!");`
  - `imprime_string(var); /* var = vetor de chars (string) */`

Passagem de Ponteiros

- **Exemplo:**

```
#include <stdio.h>

void imprime_vetor(int *v, int tamanho);

void main(void){
       int vet[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
       imprime_vetor(vet, 10);
}

void imprime_vetor(int *v, int tamanho) {
   int cont;
   for (cont=0; cont<tamanho; cont++) printf("%d\n", v[cont]);
}
```

Outros Tópicos sobre Ponteiros

- Ponteiros para Alocação Dinâmica de Memória
- Funções que Retornam Ponteiros
- Matrizes de Ponteiros
- Ponteiros para Ponteiros
- Ponteiros para Funções
- ...
- Para saber mais ...
  - (Schildt, 1997)
  - (Damas, 2007)

Exercícios

- Explique o que será apresentado como resultado do seguinte código em C:

```
#include <stdio.h>
void main(void){
   int A=1, B=2, C=3, *P1, *P2, *P3;
   P1 = &B;
   P2 = P1;
   P3 = &C;
   *P2 = 5;
   C = *P1;
   P1 = &A;
   *P1 = *P3;
   printf("%d\n%d\n%d",A, B, C);
}
```


Exercícios

- Definir um tipo de registro (struct) com três campos: um campo numérico real, um campo caractere e um campo dado por um vetor de inteiros
- Declarar um registro do tipo acima
- Declarar um ponteiro para esse tipo de estrutura e apontá-lo para o registro declarado no item anterior
- Atribuir valores aos campos do registro utilizando **o ponteiro** (ou seja, o operador seta, ao invés do operador ponto)

Exercícios

- Faça uma função que recebe como parâmetros três ponteiros para vetores de inteiros, todos do mesmo tamanho, tamanho este que também é passado como parâmetro para a função. Essa função deve fazer com que cada célula do terceiro vetor se torne igual à soma das células correspondentes dos outros dois

Exercícios

- Faça uma função que recebe como parâmetro uma string (vetor de caracteres) e o respectivo número de caracteres válidos (tamanho da string, sem contar o \0) como parâmetros. Essa função deve "inverter" a string, ou seja, fazer com que ela apareça de trás para frente caso seja escrita.

Bibliografia

- Schildt, H. "C Completo e Total", 3a. Edição, Pearson, 1997.
- Damas, L. "Linguagem C", 10a. Edição, LTC, 2007