Programação de Computadores II

Cap. 4 – Ponteiros (p.45)

Livro: Waldemar Celes, Renato Cerqueira, José Lucas Rangel. Introdução a Estruturas de Dados, Editora Campus (2004)

Slides adaptados dos originais dos profs.: Marco Antonio Casanova e Marcelo Gattass (PUC-Rio)

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

Capítulo 4 – Ponteiros e Endereços de Variáveis

Tópicos

- Ponteiros
- Pré-processador e macros

Ponteiros

```
int main ( void )
{
   int a=4;
   int *p;
   p = &a;

   *p = 2;
   printf(" %d ", a);
   return 0;
}

imprime o valor 2
```


Ponteiros

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

imprime o valor 2
```


Ponteiros: cuidados

```
int main ( void )
{
   int a, b, *p;
   a = 2;
   *p = 3;
   b = a + (*p);
   printf(" %d ", b);
   return 0;
}
```

- erro na atribuição *p = 3
  - utiliza a memória apontada por p para armazenar o valor 3, sem que p tivesse sido inicializada, logo
  - armazena 3 num espaço de memória desconhecido

Funções que mudam valores de variáveis de outras

```
#include <stdio.h>

void troca(int a, int b);

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

void troca(int a, int b) {
    int tmp=b;
    b=a;
    a=tmp;
}
```

```
a=10  b=20
Press any key to continue
```


Funções que mudam valores de variáveis de outras

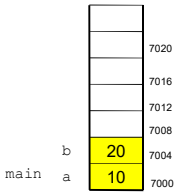
```
#include <stdio.h>

void troca(int a, int b);

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

void troca(int a, int b) {
int tmp=b;
b=a;
a=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

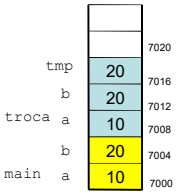
```
#include <stdio.h>

void troca(int a, int b);

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

void troca(int a, int b) {
int tmp=b;
b=a;
a=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

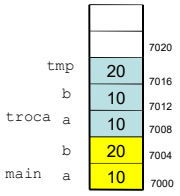
```
#include <stdio.h>

void troca(int a, int b);

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

void troca(int a, int b) {
int tmp=b;
b=a;
a=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

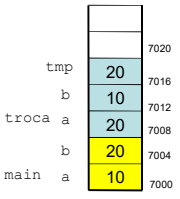
```
#include <stdio.h>

void troca(int a, int b);

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

void troca(int a, int b) {
int tmp=b;
b=a;
a=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

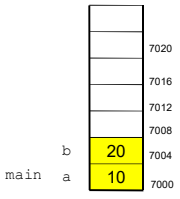
```
#include <stdio.h>

void troca(int a, int b);

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

void troca(int a, int b) {
int tmp=b;
b=a;
a=tmp;
}
```

Pilha de memória



Ponteiros

- Passagem de ponteiros para funções:
  - função g chama 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

Funções que mudam valores de variáveis de outras (passagem por referência)

```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
    int tmp=*pb;
    *pb=*pa;
    *pa=tmp;
}
```

```
a=20  b=10
Press any key to continue
```


Funções que mudam valores de variáveis de outras

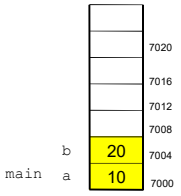
```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
int tmp=*pb;
*pb=*pa;
*pa=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

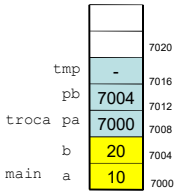
```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
int tmp=*pb;
*pb=*pa;
*pa=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

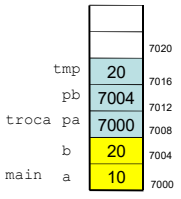
```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
int tmp=*pb;
*pb=*pa;
*pa=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

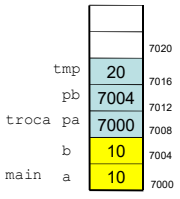
```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
int tmp=*pb;
*pb=*pa;
*pa=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

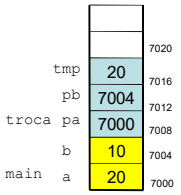
```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
int tmp=*pb;
*pb=*pa;
*pa=tmp;
}
```

Pilha de memória



Funções que mudam valores de variáveis de outras

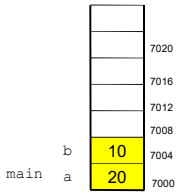
```
#include <stdio.h>

void troca(int *pa, int *pb);

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

void troca(int *pa, int *pb) {
int tmp=*pb;
*pb=*pa;
*pa=tmp;
}
```

Pilha de memória

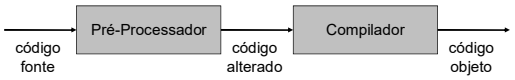



```
a=20  b=10
Press any key to continue
```


Pré-processador e macros

Pré-processador e macros

- Pré-processador:
  - reconhece determinadas diretivas
  - altera o código antes de enviá-lo ao compilador



Pré-processador e macros

- #include *nome-do-arquivo*
  - pré-processador substitui o "include" pelo corpo do arquivo especificado:
    - o texto do arquivo passa a fazer parte do código fonte
    - nome do arquivo entre aspas ("*nome-do-arquivo*"):
      - pré-processador procura o arquivo primeiro no diretório local (em geral denominado diretório de trabalho)
      - caso não o encontre, o procura nos diretórios de *include* especificados para compilação
    - nome do arquivo entre os sinais de menor e maior (<*arquivo*>):
      - pré-processador não procura o arquivo no diretório local (os arquivos da biblioteca padrão de C devem ser incluídos com <>)

Pré-processador e macros

- Diretiva de definição para representar constantes:
  - fortemente recomendável para uniformidade e clareza do código
  - exemplo:
    - função para calcular a área de um círculo
      - antes da compilação, toda ocorrência de PI (desde que não envolvida por aspas) será trocada pelo número 3.14159F

```
#define PI  3.14159F
float area (float r)
{
   float a = PI * r * r;
   return a;
}
```

Pré-processador e macros

- Macro:
  - diretiva de definição com parâmetros

```
#define MAX(a,b)    ((a) > (b) ? (a) : (b))

o pré-processador substituirá o trecho de código:

v = 4.5;
c = MAX(v, 3.0);

por:

v = 4.5;
c = ((v) > (3.0) ? (v) : (3.0));
```


Pré-processador e macros

- Cuidados na definição de macros:
  - erro de sintaxe pode ser difícil de ser detectado
  - compilador indicará erro na linha em que se utiliza a macro e não na linha de definição da macro
    (onde efetivamente encontra-se o erro)
  - **regra básica para a definição de macros:**
    - **envolva cada parâmetro, além da macro como um todo, entre parênteses**

Pré-processador e macros

```
#include <stdio.h>
#define DIF(a,b)   a - b
int main (void)
{
   printf(" %d ", 4 * DIF(5,3));
   return 0;
}
```

- problema:
  - o resultado impresso é 17 e não 8 pois o compilador processa
    *printf(" %d ", 4 * 5 - 3)*
    e a multiplicação tem precedência sobre a subtração
- solução:
  - parênteses envolvendo a macro:
    #define DIF(a,b) ((a) – (b))

Pré-processador e macros

```
#include <stdio.h>
#define PROD(a,b)   (a * b)
int main (void)
{
   printf(" %d ", PROD(3+4, 2));
   return 0;
}
```

- problema:
  - o resultado é 11 e não 14
- solução:
  - parênteses envolvendo a macro:

    #define PROD(a,b) ((a) * (b))

Resumo

Operador unário **&** (“endereço de”)

Operador unário ***** (“conteúdo de”)

```
#include nome-do-arquivo

#define código
```