

22. Arranjos e sub-rotinas

DIM0321

2015.1

Outline

- 1 Introdução
- 2 Arranjos e ponteiros
- 3 Arranjos como parâmetros
- 4 Preprocessador

- 1 Introdução
- 2 Arranjos e ponteiros
- 3 Arranjos como parâmetros
- 4 Preprocessador

Objetivos

Principal

Apresentar um mecanismo para passar um **conjunto de elementos** (i.e. arranjo) através de um parâmetro de sub-rotina.

Lembretes

Parâmetros de função \approx variáveis locais.

- Inicializadas com a cópia do valor dos argumentos da chamada
- Que acontece com a passagem de um arranjo como parâmetro ?

Exercício

- Qual é a saída do seguinte programa ?

```
1 #include <stdio.h>
2
3 void misterio(int a[8])
4 {
5     int i;
6     for( i = 1; i < 8; ++i )
7         a[i] = a[i] + a[i - 1];
8 }
9
10 int main(void)
11 {
12     int v[8] = {1, 1, 1, 1, 1, 1, 1, 1};
13     int i;
14     misterio(v);
15     for (i = 0; i < 8; ++i) printf("%i, ", v[i]);
16     return 0;
17 }
```

1 Introdução

2 Arranjos e ponteiros

3 Arranjos como parâmetros

4 Preprocessador

Lembretes

- Arranjo = blocos contíguos de memória que podem armazenar elementos de um determinado tipo de dado.
 - Um bloco = um elemento do arranjo
 - Um bloco ≥ 1 byte
- Precisamos apenas de:
 - um ponteiro inicial,
 - um deslocamento para acessar os elementos do arranjo.

Exemplo reescrito

```
1 #include <stdio.h>
2
3 void misterio(int *a, int tamanho)
4 {
5     int i;
6     for (i = 1; i < tamanho; ++i) a[i] = a[i] + a[i - 1];
7 }
8
9
10 int main(void)
11 {
12     int v[8] = { 1, 1, 1, 1, 1, 1, 1, 1 };
13     int i;
14     misterio(v, 8);
15     for (i = 0; i < 8; ++i) printf("%i, ", v[i]);
16     return 0;
17 }
```

Observação

- Dominar essa forma de usar arranjos / blocos de memória \Rightarrow
aritmética de ponteiros

Aritmética de ponteiros

Lembretes

- $ptr + i = \text{endereço de } ptr + i * \text{sizeof}(\text{tipo de dado do } ptr)$
- $ptr - i = \text{endereço de } ptr - i * \text{sizeof}(\text{tipo de dado do } ptr)$

Exemplo

```
1 #include <stdio.h>
2
3 void misterio(int *a)
4 {
5     int i;
6     for (i = 1; i < 8; ++i)
7         /* a[i] == *(a + i) */
8         *(a + i) = *(a + i) + *(a + i - 1);
9 }
10
11 int main(void)
12 {
13     int v[8] = { 1, 1, 1, 1, 1, 1, 1, 1 };
14     int i;
15     misterio(v);
16     for (i = 0; i < 8; ++i) printf("%i, ", v[i]);
17     return 0;
18 }
```

Exemplo

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i, v[5];
6     printf("%u\n", sizeof(int));
7     for (i = 0; i < 5; ++i) printf("v[%i]: %p\n", i, &v[i]);
8     printf("\nv: %p", v);
9     printf("\nv[3]: %p", v + 3);
10    return 0;
11 }
```

- 1 Introdução
- 2 Arranjos e ponteiros
- 3 Arranjos como parâmetros**
- 4 Preprocessador

Elementos gerais

Observações

- Quando uma variável arranjo (ponteiro) é passada como parâmetro, ela não possui informação sobre o seu tamanho.
- Para que uma sub-rotina possa manipular um arranjo de tamanho qualquer, este tamanho também deve ser informado.

Exemplo: uso com tamanho

```
1 #include <stdio.h>
2
3 void incrementa_arranjo(int *v, int tamanho)
4 {
5     int i;
6     for (i = 0; i < tamanho; ++i) v[i]++; // v[i] = v[i] + 1;
7 }
8
9 void intermediaria()
10 {
11     int i, j;
12     scanf("%u", &i);
13     int v[i];
14     for (j = 0; j < i; j++) v[j] = j + i / j * 2;
15     incrementa_arranjo(v, i);
16 }
17
18 int main(void)
19 {
20     int x[4] = { 0, 1, 2, 3 };
21     int y[5] = { 7, 6, 5, 4, 3 };
22     int z[2] = { 1, 2 };
23     incrementa_arranjo(x, 4);
24     incrementa_arranjo(y, 5);
25     incrementa_arranjo(z, 2);
26     intermediaria();
27     return 0;
28 }
```

Exemplo : inverter elementos de um arranjo

```
1 void inverte_arranjo(float *v, int tamanho)
2 {
3     int i, j;
4     float tmp;
5     for (i = 0, j = tamanho - 1; i < j; ++i, --j) {
6         tmp = v[i];
7         v[i] = v[j];
8         v[j] = tmp;
9     }
10 }
```

Exercício

- Qual é a saída do seguinte programa ?

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     unsigned long v[10];
6     unsigned long *p;
7     p = v;
8     v[0] = v;
9
10     printf("v[0] = %i, p = %i, %u, %u\n", v[0], p, sizeof(p), sizeof(v));
11     if (p == *v) printf("1\n");
12     if (*p == &v[0]) printf("2\n");
13     if (p == &v[0]) printf("3\n");
14     if (*p == v[0]) printf("4\n");
15     return 0;
16 }
```

- 1 Introdução
- 2 Arranjos e ponteiros
- 3 Arranjos como parâmetros
- 4 Preprocessador

Como funciona ?

- O pré-processo é a 1a. etapa separada da compilação.
- Vai interpretar várias diretivas de:
 - ▶ inclusão
 - ▶ definição
 - ▶ condição

Inclusão de arquivo

Comando

- ① `#include "filename"`
- ② `#include <filename>`

Funcionamento

- A linha é substituída por o arquivo
- O efeito pode ser visto com `gcc -E <nomearquivo.c>`
- Se `filename` for entre aspas ("")
 - ▶ o arquivo `filename` é procurado primeiro onde o programa foi achado
 - ▶ se não achado, ou se `filename` for entre <>, a busca do arquivo segue outras regras, definidas por seu compilador (vai procurar geralmente no `/usr/include` no Linux).

Definição de macros

Comando

```
#define name replacement_text  
#define MESES 12
```

Inclusão condicional

De um cabeçalho, sem repetição

```
1 #if !defined(HDR)  
2 #define HDR  
3 /* contents of hdr.h go here */  
4 #endif
```

De cabeçalhos diferentes em função de outros dados

```
1 #if SYSTEM == SYSV  
2 #define HDR "sysv.h"  
3 #elif SYSTEM == BSD  
4 #define HDR "bsd.h"  
5 #elif SYSTEM == MSDOS  
6 #define HDR "msdos.h"  
7 #else  
8 #define HDR "default.h"  
9 #endif  
10 #include HDR
```

Precisões

[KR88] 5, 4.11

[Bac13] 9.2, 10

-  André Backes, *Linguagem C completa e descomplicada*, Elsevier, 2013.
-  Brian W. Kernighan and Dennis M. Ritchie, *The (ANSI) C Programming Language*, 2nd ed., Prentice Hall Professional Technical Reference, 1988.



<http://dimap.ufrn.br/~richard/dim0321>