# 22. Arranjos e sub-rotinas

DIM0321

2015.1

# Outline

# 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 ?

```c
#include <stdio.h>

void misterio(int a[8])
{
    int i;
    for( i = 1; i < 8; ++i )
        a[i] = a[i] + a[i - 1];
}

int main(void)
{
    int v[8] = {1, 1, 1, 1, 1, 1, 1, 1};
    int i;
    misterio(v);
    for (i = 0; i < 8; ++i) printf("%i, ", v[i]);
    return 0;
}
```

# Lembretes

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

# Exemplo reescrito

```c
#include <stdio.h>

void misterio(int *a, int tamanho)
{
    int i;
    for (i = 1; i < tamanho; ++i) a[i] = a[i] + a[i - 1];
}


int main(void)
{
    int v[8] = { 1, 1, 1, 1, 1, 1, 1, 1 };
    int i;
    misterio(v, 8);
    for (i = 0; i < 8; ++i) printf("%i, ", v[i]);
    return 0;
}
```

## Observação

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

# Aritmética de ponteiros

## Lembretes

- `ptr + i` = endereço de `ptr + i * sizeof (<tipo de dado do ptr >)`
- `ptr - i` = endereço de `ptr - i * sizeof (<tipo de dado do ptr >)`

## Exemplo

```c
#include <stdio.h>

void misterio(int *a)
{
    int i;
    for (i = 1; i < 8; ++i)
      /* a[i] == *(a + i) */
      *(a + i) = *(a + i) + *(a + i - 1);
}

int main(void)
{
    int v[8] = { 1, 1, 1, 1, 1, 1, 1, 1 };
    int i;
    misterio(v);
    for (i = 0; i < 8; ++i) printf("%i, ", v[i]);
    return 0;
}
```

# Exemplo

```c
#include <stdio.h>

int main(void)
{
    int i, v[5];
    printf("%u\n", sizeof(int));
    for (i = 0; i < 5; ++i) printf("v[%i]: %p\n", i, &v[i]);
    printf("\nv: %p", v);
    printf("\nv[3]: %p", v + 3);
    return 0;
}
```

# 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

```c
#include <stdio.h>

void incrementa_arranjo(int *v, int tamanho)
{
    int i;
    for (i = 0; i < tamanho; ++i) v[i]++; // v[i] = v[i] + 1;
}

void intermediaria()
{
    int i, j;
    scanf("%u", &i);
    int v[i];
    for (j = 0; j < i; j++) v[j] = j + i / j * 2;
    incrementa_arranjo(v, i);
}

int main(void)
{
    int x[4] = { 0, 1, 2, 3 };
    int y[5] = { 7, 6, 5, 4, 3 };
    int z[2] = { 1, 2 };
    incrementa_arranjo(x, 4);
    incrementa_arranjo(y, 5);
    incrementa_arranjo(z, 2);
    intermediaria();
    return 0;
}
```

# 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 ?

```c
#include <stdio.h>

int main(void)
{
    unsigned long v[10];
    unsigned long *p;
    p = v;
    v[0] = v;

    printf("v[0] = %i, p = %i, %u, %u\n", v[0], p, sizeof(p), sizeof(v));
    if (p == *v) printf("1\n");
    if (*p == &v[0]) printf("2\n");
    if (p == &v[0]) printf("3\n");
    if (*p == v[0]) printf("4\n");
    return 0;
}
```

# Como funciona ?

- O preprocesso 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

```
#if !defined(HDR)
#define HDR
/* contents of hdr.hgo here */
#endif
```

## De cabeçalhos diferentes em função de outros dados

```
#if SYSTEM == SYSV
#define HDR "sysv.h"
#elif SYSTEM == BSD
#define HDR "bsd.h"
#elif SYSTEM == MSDOS
#define HDR "msdos.h"
#else
#define HDR "default.h"
#endif
#include HDR
```

# Referências

## 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.

# Perguntas ?


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