

# 10. Subrotinas

DIM0321

2015.1

# Outline

1 Introdução

2 Subrotinas em C

1 Introdução

2 Subrotinas em C

# Motivação

## Observação

- Os programas escritos até então estão inseridos na função principal `main`
- Já usamos recursos externos (parte da biblioteca padrão / arquivos cabeçalhos)
- Recursos externos = componentes do código

## Modularização

Numa aplicação real, tarefas repetitivas ou que podem ser reusadas são organizadas em módulos

- inverter um matriz
- ordenar um vetor (de nome, de número)

# Sub-rotinas

## Objetivo

- propõem a prestação de um serviço preciso
- são relativamente independentes do resto do programa

## Estruturar o código

- Subrotinas encontram-se num ponto pré-definido do programa
- Podem ser chamadas a partir de vários pontos do programa
- Podem ser modificadas de maneira relativamente independente, desde que o serviço prestado continue o mesmo

Elas são uma maneira de introduzir uma separação lógica entre a implementação/ algoritmo usado (o corpo da sub-rotina) e o resultado.

1 Introdução

2 Subrotinas em C

# Elementos de sub-rotina

## Partes

- Nome da subrotina
- Tipo de retorno
- Parâmetros
- Corpo
- Valor de retorno

## Funções

- Em C, toda subrotina é uma função
- Procedimento  $\approx$  função com tipo de retorno void

# Declaração

## Definição (Declaração)

- A interface = declaração de prestação de serviços
- Descreve como a sub-rotina deve ser chamada. Ela é **necessária**.

```
1| tipo_retorno nome_funcao(tipo1 par1, tipo2 par2, ..., tipon parn);
```

## Exemplo

```
1| dummy(); // declaração mínima
2|
3| double pow(double x, double y);
4| int printf(const char *format, ...);
```

# Definição

## Definição (Definição)

- A definição da sub-rotina é a etapa de **implementação**
- Definimos o corpo da sub-rotina
- Pode ocorrer em qualquer lugar do código apos a declaração

```
1 | tipo_retorno nome_funcao(tipo1 par1, tipo2 par2, ..., tipon parn)
2 | {
3 |     // declaracoes e comandos
4 | }
```

## Elementos

- Repetição da declaração
- Variáveis locais
- Instruções de retorno (se necessário)

# Exemplos

## V1

```
1 #include <stdio.h>
2
3 void print_int(int d)
4 {
5     printf("Int: %d\n", d);
6 }
7
8 int square_int(int d);
9
10 int main(void)
11 {
12     int n;
13     printf("Enter an int: ");
14     scanf("%d", &n);
15     print_int(square_int(n));
16     return 0;
17 }
18
19 int square_int(int d)
20 {
21     int sq;
22     sq = d * d;
23     return sq;
24 }
```

## V2

```
1 #include <stdio.h>
2
3 void print_int(int d)
4 {
5     printf("Int: %d\n", d);
6 }
7
8 int square_int(int d)
9 {
10     return d * d;
11 }
12
13
14 int main(void)
15 {
16     int n;
17     printf("Enter an int: ");
18     scanf("%d", &n);
19     print_int(square_int(n));
20     return 0;
21 }
```

# Variáveis globais

- Até agora, só usamos variáveis definidas no início de um bloco ou dentro de um laço `for`.
- Podemos também compartilhar variáveis entre componentes, através variáveis globais declaradas **no mesmo nível** que as funções.

## Getter/setter

```
1 int glob;
2 void setglob(int v);
3 int getglob(void);
4
5 void setglob(int v) { glob = v; }
6 int getglob(void) { return glob; }
```

## Aviso

- Não é aconselhado usar variáveis globais **sem necessidade**, pois elas quebram as separações entre componentes.

# Exemplo (K & R)

```
1 #define BUFSIZE 100
2
3 char buf[BUFSIZE];
4 int bufp = 0;
5
6 /* buffer for ungetch */
7 /* next free position in buf */
8 int getch(void) /* get a (possibly pushed-back) character */
9 {
10     return (bufp > 0) ? buf[--bufp] : getchar();
11 }
12
13
14 void ungetch(int c)
15 /* push character back on input */
16 {
17     if (bufp >= BUFSIZE)
18     printf("ungetch: too many characters\n");
19     else
20     buf[bufp++] = c;
21 }
```

# Escopo

## Definição

O **escopo** de um nome é a parte do código na qual ele pode ser usado.

## Exemplo

- Para variáveis locais, é o corpo da função.
- Para variáveis declaradas no início de um bloco, é o bloco mesmo.

# Variáveis globais : declaração e definição

## Declaração + definição

```
1 | int sp;  
2 | double val[MAXVAL];
```

- A variável é criada e inicializada
- O espaço de memória é alocado

## Declaração só

```
1 | extern int sp;  
2 | extern double val[MAXVAL];
```

- A variável é definida fora (por exemplo, num outro arquivo).
- Ela não é criada
- O espaço de memória é alocado fora.

# Exemplo

- Qual é o resultado retornado por essa função ?

```
1 int x = 5;  
2 int f() {  
3     int x = 3;  
4     {  
5         extern int x;  
6         return x;  
7     }  
8 }
```

# Referências

## Precisões

[KR88] Seções 4.1 – 4.5

[Bac13] 9.1



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>