Programação Procedimental
GBC014 – 2015/1
Prof. Renan Cattelan – www.facom.ufu.br/~renan

Prática 8

Funções

Funções

- Funções são blocos de código que podem ser nomeados e chamados de dentro de um programa
  - Ex.: printf(), scanf(), strcpy(), pow(), system(), main()
- Funções permitem
  - Estruturação
  - Reuso

Funções

- Declaração

```
tipo_retornado nome_função(parâmetros){
   conjunto de declarações e comandos
}
```

  - Parâmetros: lista de variáveis + tipos
  - Corpo da função: declarações + comandos
  - Valor de retorno: tipo pré-definido ou estrutura

Exemplo de função

```
01  int maior(int x, int y){
02      if(x > y)
03          return x;
04      else
05          return y;
06  }
```

Declaração de funções

- Funções devem ser definidas ou declaradas antes de serem utilizadas, ou seja, antes da cláusula main
- A definição (protótipo) apenas indica a existência da função:

tipo_retornado nome_função(parâmetros);

- Desse modo ela pode ser declarada após a cláusula main()

```c
#include <stdio.h>
#include <stdlib.h>

int Square (int a){
  return (a*a);
}

int main(){
  int n1,n2;
  printf("Entre com um numero: ");
  scanf("%d", &n1);
  n2 = Square(n1);
  printf("O seu quadrado vale: %d\n", n2);
  system("pause");
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
//protótipo da função
int Square (int a);

int main(){
  int n1,n2;
  printf("Entre com um numero: ");
  scanf("%d", &n1);
  n2 = Square(n1);
  printf("O seu quadrado vale: %d\n", n2);
  system("pause");
  return 0;
}

int Square (int a){
  return (a*a);
}
```

Lista 7

3. Faça uma função que receba por parâmetro dois valores X e Z. Calcule e retorne o resultado de X^Z para o programa principal. Atenção não utilize nenhuma função pronta de exponenciação.

```c
#include<stdio.h>
#include<stdlib.h>

int exponenciacao(int x, int z) {
    int resultado = x, i;
    for (i=1; i<z; i++) {
        resultado = resultado * x;
    }
    return resultado;
}

int main(void)
{
    int x, z, res;
    printf("Digite o valor de x e z: ");
    scanf("%d %d", &x, &z);

    res = exponenciacao(x, z);

    printf("\nO resultado eh: %d\n", res);

    system("pause");
    return 0;
}

```

Escopo de variáveis

- Local
  - Validade dentro do bloco no qual são declaradas
- Global
  - Declaradas fora de todas as funções do programa
  - São conhecidas e podem ser alteradas por todas as funções do programa

Lista 7

14. Faça um algoritmo que receba um número inteiro positivo n e calcule:

    (a) n!

    (b) Somatório de 1 até n

    Obs.: Crie uma função para cada letra.

```c
#include<stdio.h>

int fatorial(int n) {
    if (n < 0) return -1;
    int i, resultado = 1;
    for (i=2; i<=n; i++) {
        resultado = resultado * i;
    }
    return resultado;
}

int somatorio(int n) {
    if (n < 1) return n;
    int i, soma = 0;
    for (i=1; i<=n; i++) {
        soma = soma + i;
    }
    return soma;
}
```

```
20
21  int main(void)
22  {
23      int n, f, s;
24      printf("Digite um numero: ");
25      scanf("%d", &n);
26
27      f = fatorial(n);
28      printf("\nO fatorial de %d eh: %d", n, f);
29
30      s = somatorio(n);
31      printf("\nO somatorio de %d eh: %d\n", n, s);
32
33      system("pause");
34      return 0;
35  }
```

Passagem de parâmetros

- Por valor
  - Uma cópia do valor do parâmetro é feita e passada para a função
  - Mesmo que esse valor mude dentro da função, nada acontece com o valor de fora da função
  - Padrão da Linguagem C
- Por referência
  - A referência (seu endereço na memória) da variável é passada à função
  - Qualquer alteração que a variável sofra dentro da função será refletida fora da função
  - Arrays são sempre passados por referência

Passagem por valor

```
01  include <stdio.h>
02  include <stdlib.h>
03
04  void soma _ mais _ um(int n){
05    n = n + 1;
06    printf("Dentro da funcao: x = %d\n",n);
07  }
08
09  int main(){
10    int x = 5;
11    printf("Antes da funcao: x = %d\n",x);
12    soma _ mais _ um(x);
13    printf("Depois da funcao: x = %d\n",x);
14    system("pause");
15    return 0;
16  }
```

```
Saída   Antes da funcao: x = 5
        Dentro da funcao: x = 6
        Depois da funcao: x = 5
```

Passagem por referência

```
01      include <stdio.h>
02      include <stdlib.h>
03
04      void soma _ mais _ um(int *n){
05        *n = *n + 1;
06         printf("Dentro da funcao: x = %d\n",*n);
07      }
08
09      int main(){
10        int x = 5;
11        printf("Antes da funcao: x = %d\n",x);
12        soma _ mais _ um(&x);
13        printf("Depois da funcao: x = %d\n",x);
14        system("pause");
15        return 0;
16      }
```

```
Saída    Antes da funcao: x = 5
         Dentro da funcao: x = 6
         Depois da funcao: x = 6
```

Recursividade

recursividade
re.cur.si.vi.da.de
sf (***recursivo+i+dade***) ***Ling*** Propriedade sintática pela qual um elemento pode aparecer um número infinito de vezes numa derivação, introduzido sempre pela mesma regra. ***Ex: O filho da irmã da mãe de José***.

- O termo é vastamente aplicado em Computação
  - Ex.: GNU/Linux
  - **G**NU is **N**ot **U**nix
    - Acrônimo recursivo

Recursão em Linguagens de Programação

- Uma **função recursiva** é uma função que chama a si mesma

```
int fatorial(int n) {
    if (n <= 1)
        return 1;
    return n * fatorial(n-1);
}
```

Recursão em Linguagens de Programação

- Uma **função recursiva** é uma função que chama a si mesma

```
int fatorial(int n) {
    if (n <= 1)
        return 1;
    return n * fatorial(n-1);
}
```

Chamada recursiva

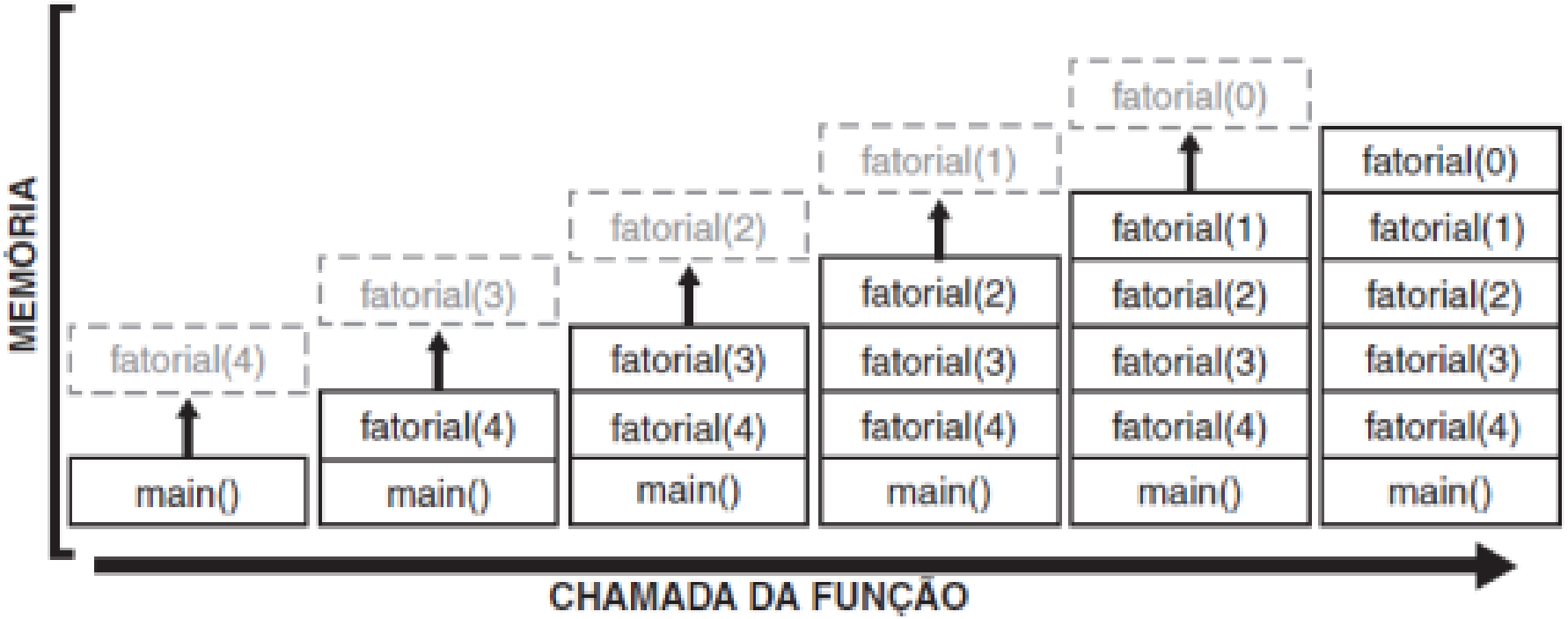
Recursão em Linguagens de Programação

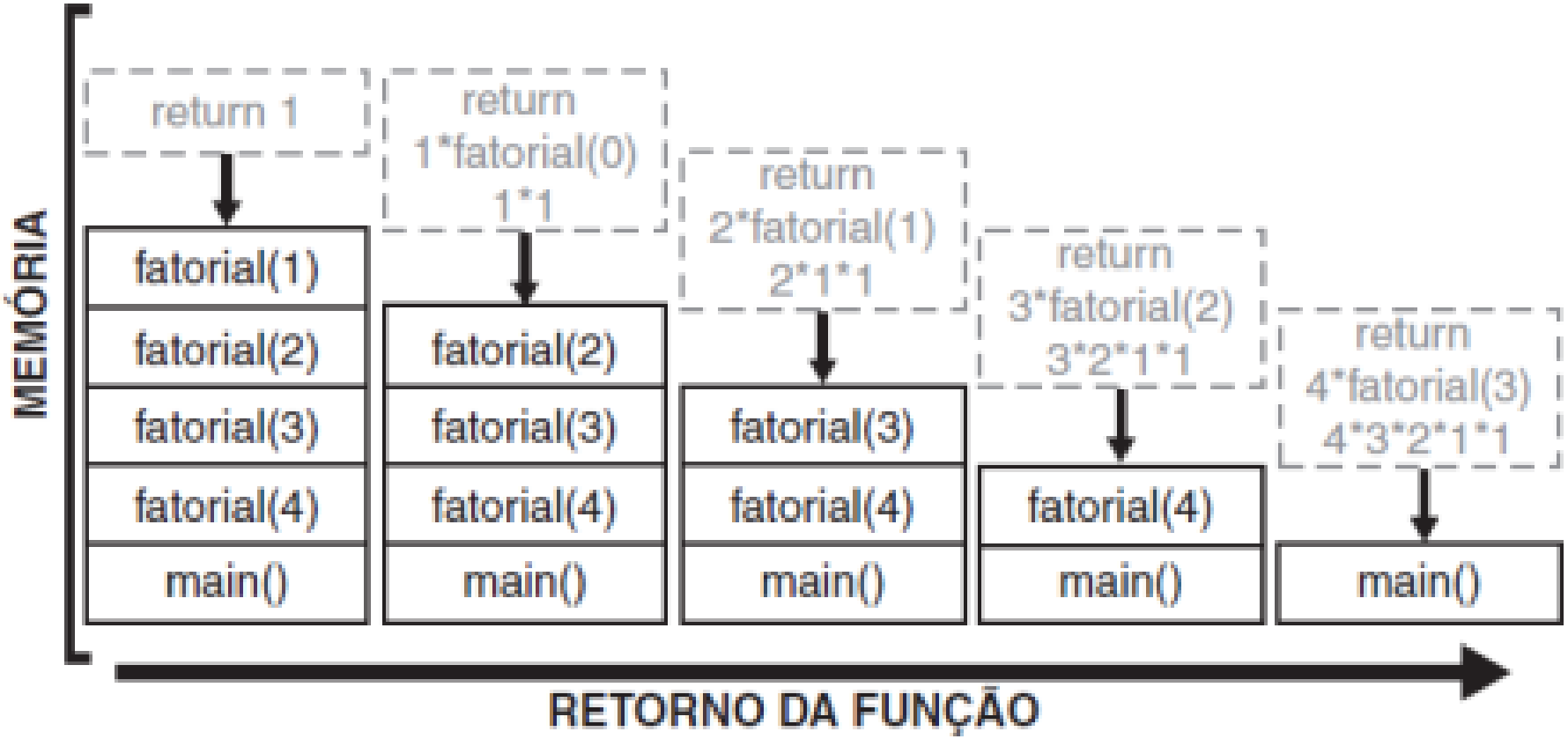
- Uma **função recursiva** é uma função que chama a si mesma

```
int fatorial(int n) {
    if (n <= 1)
        return 1;
    return n * fatorial(n-1);
}
```

Critério de parada: Determina quando a função deverá parar de chamar a si mesma

```c
#include<stdio.h>

int fatorial(int n) {
    if (n <= 1)
        return 1;
    return n * fatorial(n-1);
}

int main(void)
{
    int n;
    printf("Digite um numero: ");
    scanf("%d", &n);

    printf("\nO fatorial de %d eh: %d\n", n, fatorial(n));

    system("pause");
    return 0;
}
```


MEMÓRIA
fatorial(4)
fatorial(3)
fatorial(2)
fatorial(1)
fatorial(0)
main()
fatorial(4)
fatorial(3)
fatorial(2)
fatorial(1)
main()
fatorial(4)
fatorial(3)
fatorial(2)
main()
fatorial(4)
fatorial(3)
main()
fatorial(4)
main()
fatorial(0)
fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)
main()
CHAMADA DA FUNÇÃO


MEMÓRIA
return 1
return
1*fatorial(0)
1*1
return
2*fatorial(1)
2*1*1
return
3*fatorial(2)
3*2*1*1
return
4*fatorial(3)
4*3*2*1*1
fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)
main()
fatorial(2)
fatorial(3)
fatorial(4)
main()
fatorial(3)
fatorial(4)
main()
fatorial(4)
main()
main()
RETORNO DA FUNÇÃO

Versão iterativa

- Dada uma função recursiva, sempre é possível escrever uma função equivalente, sem recursão (função iterativa)

```
int fatorial(int n) {
    if (n < 0) return -1;
    int i, resultado = 1;
    for (i=2; i<=n; i++) {
        resultado = resultado * i;
    }
    return resultado;
}
```

Exercício sobre recursão

- Escreva uma função recursiva eficiente que receba dois inteiros positivos k e n e calcule k^n

```c
#include<stdio.h>
#include<stdlib.h>

int recursive_pow(int k, int n) {
    if (n == 0) return 1;
    return k*recursive_pow(k,n-1);
}

int main(void)
{
    int k, n;

    printf("\nDigite os valores de k e n: ");
    scanf("%d %d", &k, &n);

    printf("\nO resultado de k^n eh: %d\n", recursive_pow(k,n));

    system("pause");
    return 0;
}
```