“Se um café pela manhã não te acordar, tente rodar o seguinte em um servidor que está em produção: rm -rf --no-preserve-root /”

Funções não folha

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Funções folha

- A função apresentada na aula passada é uma **função folha**

```
int leaf_example(int g, int h, int i, int j){
      int f;
      f = (g+h) - (i+j);
      return f;
}
```

  - Realiza sua tarefa e retorna sem chamar uma outra função
    - Seríamos mais felizes se toda função fosse folha
    - Mas nem tudo são flores (na vida de Joseph Climber, e na sua)

Funções não-folha

- Uma função que chama outra internamente para resolver o problema é uma **função não-folha**
  - Procedimentos aninhados
- A função pode **chamar outra função**, ou um **clone de si mesma (recursão)**
  - Os problemas são **os mesmos em ambos os casos**

Exemplo

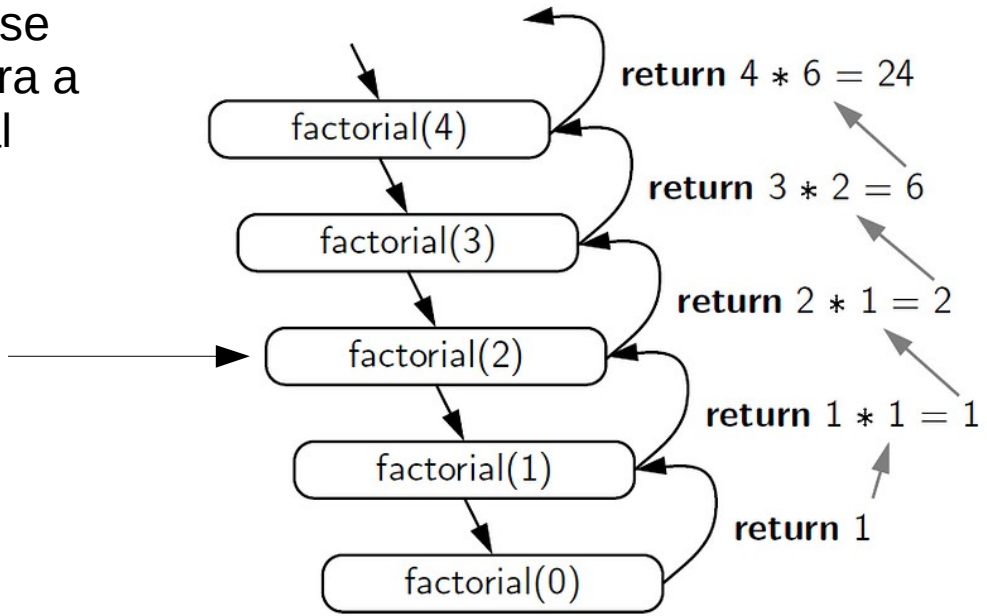
- Considere a função que calcula o fatorial recursivamente
  - Que problemas criamos agora a nível de linguagem de montagem e de máquina que não existiam em uma função folha?

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

Exemplo

- Considere a função que calcula o fatorial recursivamente
  - Que problemas criamos agora a nível de linguagem de montagem e de máquina que não existiam em uma função folha?
    - Os valores dos registradores podem se perder
    - O endereço de retorno em $ra vai se perder, e não saberemos voltar para a função original que chamou fatorial

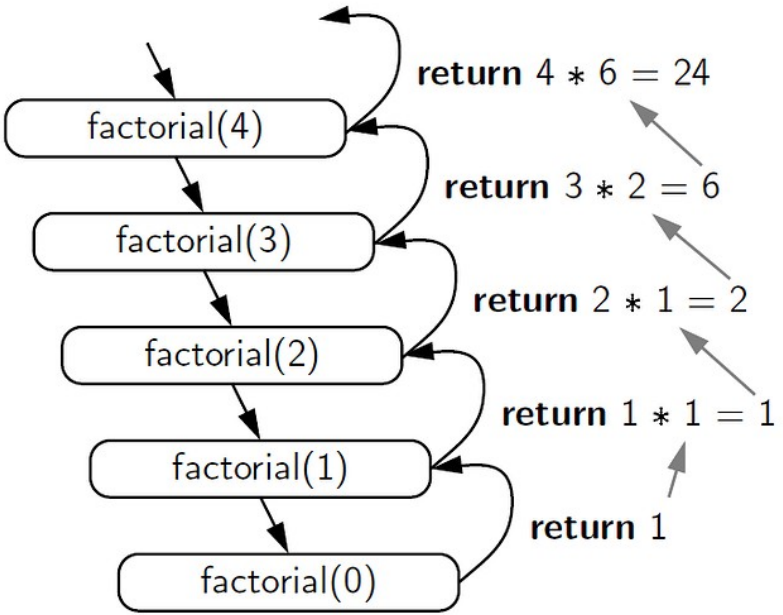
Cada chamada de fatorial deveria ter seus próprios registradores (ex.: s0, s1) e seu próprio retorno



fonte: discuss.codecademy.com/t/5-factorial/67584/8

Funções não-folha

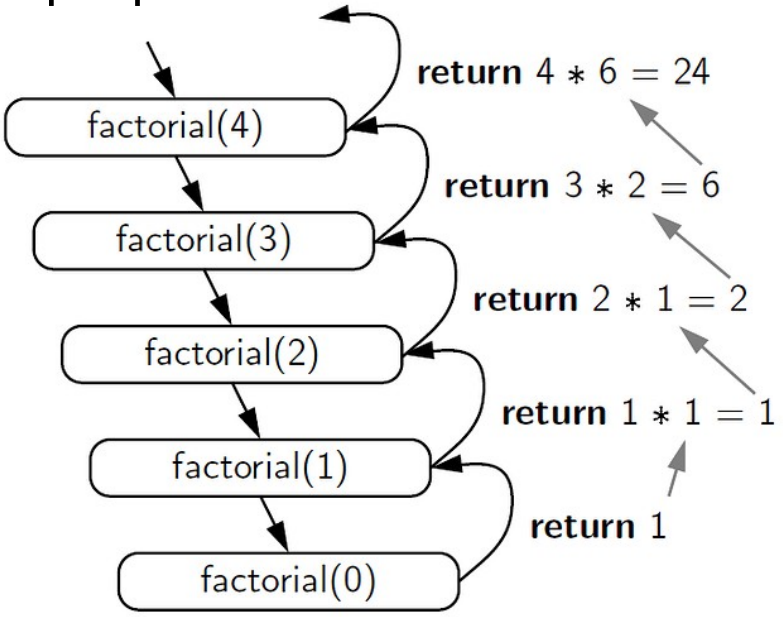
- Cada chamada de fatorial deveria ter seus próprios registradores (ex.: s0, s1) e seu próprio retorno
  - Como podemos resolver?



fonte: discuss.codecademy.com/t/5-factorial/67584/8

Funções não-folha

- Cada chamada de fatorial deveria ter seus próprios registradores (ex.: s0, s1) e seu próprio retorno
  - Como podemos resolver?
  - Podemos mais uma vez empilhar os valores que precisam ser salvos



fonte: discuss.codecademy.com/t/5-factorial/67584/8

Criação do Fatorial Recursivo

```
fatorial_rec:
        slti $t0,$a0,1                  #a0 contém o parâmetro n
        bne $t0,$zero,fatorial_0        #salta se chegamos em 0
        #continua aqui!

fatorial_0:                             #retornar o fatorial de 0
        ori $v0,$zero,1
        jr $ra
```

Aqui é para o caso não folha

Se chamamos para o fatorial de 0, simplesmente retornamos 1 sem chamar nenhuma outra função (se tornou folha)

Criação do Fatorial Recursivo

Precisamos salvar o valor de n (a0), e também saber para onde retornar. Para que nada se perca, vamos salvar na pilha.

```
fatorial_rec:
        slti $t0,$a0,1          #a0 contém o parâmetro n
        bne $t0,$zero,fatorial_0  #salta se chegamos em 0
        addi $sp, $sp, -8       #ajusta a pilha para 2 itens
        sw $a0, 0($sp)          #guarda a0 para depois
        sw $ra, 4($sp)          #guarda o endereço de retorno
        #continua
fatorial_0:                     #retornar o fatorial de 0
        ori $v0,$zero,1
        jr $ra
```

UDESC

Criação do Fatorial Recursivo

Ajusta para n-1 e chama fat(n-1)

Recarrega os valores salvos e apaga da pilha

Multiplica n*fat(n-1) e coloca o resultado em v0. Depois retorna para quem chamou

```
fatorial_rec:
        slti $t0,$a0,1          #a0 contém o parâmetro n
        bne $t0,$zero,fatorial_0  #salta se chegamos em 0
        addi $sp, $sp, -8       #ajusta a pilha para 2 itens
        sw $a0, 0($sp)          #guarda a0 para depois
        sw $ra, 4($sp)          #guarda o endereço de retorno
        addi $a0, $a0, -1
        jal fatorial_rec        #chama fatorial para n-1
        lw $a0, 0($sp)          #carrega o valor antigo de a0
        lw $ra, 4($sp)          #carrega o endereço
        addi $sp, $sp, 8        #apaga os valores da pilha
        mul $v0, $a0,$v0        #multiplica a0 pelo retorno do fatorial
        jr $ra
fatorial_0:                     #retornar o fatorial de 0
        ori $v0,$zero,1
        jr $ra
```

UDESC

Solução Completa

```
        .text
        .globl main
main:
        ori $v0,$zero,5
        syscall                 #lê do teclado
        ori $a0,$v0, 0          #coloca como argumento
        jal fatorial_rec
        ori $a0,$v0,0           #coloca o resultado como argumento
        ori $v0,$zero,1         #impressão de inteiro
        syscall                 #imprime o resultado
        j end
fatorial_rec:
        slti $t0,$a0,1          #a0 contém o parâmetro n
        bne $t0,$zero,fatorial_0 #salta se chegamos em 0
        addi $sp, $sp, -8       #ajusta a pilha para 2 itens
        sw $a0, 0($sp)          #guarda a0 para depois
        sw $ra, 4($sp)          #guarda o endereço de retorno
        addi $a0, $a0, -1
        jal fatorial_rec        #chama fatorial para n-1
        lw $a0, 0($sp)          #carrega o valor antigo de a0
        lw $ra, 4($sp)          #carrega o endereço
        addi $sp, $sp, 8        #apaga os valores da pilha
        mul $v0, $a0,$v0        #multiplica a0 pelo retorno do fatorial
        jr $ra
fatorial_0:                     #retornar o fatorial de 0
        ori $v0,$zero,1
        jr $ra
end:
        li $v0, 10              #10 em $v0 para informar que o programa terminou
        syscall                 #chama o S.O. para terminar o programa
```

Pontos a Considerar

- Muitos problemas possuem soluções mais simples quando utilizamos recursão
  - Exemplo: navegar em uma árvore binária
- Existem ainda problemas em que não há solução não-recursiva
  - Podemos utilizar loops e inserir as informações em uma pilha ou alguma outra estrutura de dados, mas no final das contas ainda estamos simulando uma recursão

Pontos a Considerar

- A recursão (ou mesmo chamada de procedimentos não folha) custa caro para o processador
  - Por quê?

Pontos a Considerar

- A recursão (ou mesmo chamada de procedimentos não folha) custa caro para o processador
  - Cada chamada exige comunicação com a memória para empilhar os dados
  - Ocupa espaço na pilha
  - Invalida a memória cache
- Sendo assim, uma solução iterativa é preferível a nível de linguagem de máquina
  - Nem sempre é possível
  - Compiladores modernos fazem o que podem para tentar eliminar recursões

Exercícios

1. Execute o fatorial recursivo no MARS passo a passo, analise e entenda as mudanças ocorridas em cada um dos registradores e nos endereços de memória.
2. Crie uma função recursiva para calcular o enésimo número da sequência de Fibonacci.

```
fib(n){
        if(n==0 || n == 1)
                return 1;
        return fib(n-1) + fib(n-2);
}
```

Exercícios

3. Escreva uma função recursiva que determine quantas vezes um dígito K ocorre em um número inteiro N. Por exemplo, o dígito 2 ocorre 3 vezes em 762021192.

   Fonte: programacaodescomplicada.files.wordpress.com/2012/10/lista-recursc3a3o.pdf

   exemplo da assinatura da função em C

```
int contaOcorrencias(int n, int k){
        //implementação aqui
}
```

4. Escreva uma função recursiva que calcula a soma dos dígitos de um valor inteiro qualquer passado como parâmetro. Exemplo: Para a entrada 36, a resposta é 3 + 6 = 9.

Exercícios

5. Escreva um programa que recebe como parâmetro um inteiro positivo qualquer, e retorna (em um inteiro) o seu equivalente em binário. Utilize recursão. Exemplo: Para 66_{10}, a resposta deve ser 1000010_2.

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**: **a Interface Hardware/Software.** 5a Edição. Elsevier Brasil, 2017.
- STALLINGS, William. **Arquitetura e organização de computadores.** 10. ed. São Paulo: Pearson Education do Brasil, 2018.
- Bob Plantz. **Introduction to Computer Organization: A Guide to X86-64 Assembly Language and GNU/Linux.** 2011.