Recursividade

Algoritmos e Programação 2 – Ciências da Computação – UFJ

Professora: Ana Paula Freitas Vilela Boaventura

Das últimas aula...

Strings
Structs
Variáveis locais e globais
Passagem de parâmetros por valor

Segue o plano

- Recursividade


Royal
FERMENTO
QUÍMICO EM PÓ
250 g
Royal

Recursividade

- Seja a função matemática

$$F(x) = 10 * x + 2$$

- Como a resolvemos?

- Vejamos a função matemática
- $F(x) = 10 * x + 2$

$$F(x) = 10 * x + 2$$
$$F(1) = 10 * 1 + 2 = 12$$
$$F(2) = 10 * 2 + 2 = 22$$
$$F(3) = 10 * 3 + 2 = 32$$

$$F(x) = 10 * x + 2$$

$$F(x) = 10 * x + 2$$
$$F(1) = 10 * 1 + 2 = 12$$
$$F(2) = 10 * 2 + 2 = 22$$
$$F(3) = 10 * 3 + 2 = 32$$

- Seja outra função matemática

$$G(x) = (10 * G(x-1)) + 2, \text{ para } x > 1, \text{ se } x=1, G(1)=1$$

- Como a resolvemos?

- $G(x) = (10 * G(x-1)) + 2$

$$G(x) = (10*(G(x-1)) + 2$$
$$G(4) = (10*(G(4-1)) + 2$$
$$G(4) = (10*(G(3)) + 2$$

para x>1

se x=1, G(1)=1.

x=4

(10*(G(3))+2

Empilhando

- $G(x) = (10 * G(x-1)) + 2$

G(3) = (10*(G(3-1)) + 2

G(3) = (10*(G(2)) + 2

x=3

(10*(G(2))+2

x=4

(10*(G(3))+2

para x>1

se x=1, G(1)=1.

Empilhando

- $G(x) = (10 * G(x-1)) + 2$

x=2 (10*(G(1))+2

x=3 (10*(G(2))+2

x=4 (10*(G(3))+2

Empilhando

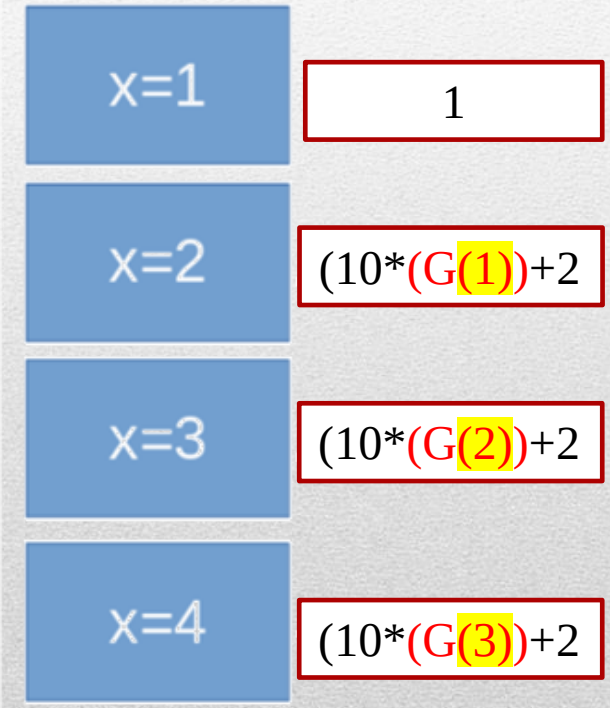
G(2) = (10*(G(2-1)) + 2

G(2) = (10*(G(1)) + 2

para x>1

se x=1, G(1)=1.

- $G(x) = (10 * G(x-1)) + 2$



Empilhando

G(1) = 1

para x>1

se x=1, G(1)=1.

- $G(x) = (10 * G(x-1)) + 2$

x=2	(10*(G(1))+2=12
x=3	(10*(G(2))+2
x=4	(10*(G(3))+2

$$G(2) = (10*(G(1)) + 2$$
$$G(2) = (10*1) + 2$$
$$G(2) = 12$$

para x>1
se x=1, G(1)=1.

Desempilhando

- $G(x) = (10 * G(x-1)) + 2$

x=3

(10*(G(2))+2 =122

x=4

(10*(G(3))+2

G(3) = (10*(G(2)) + 2

G(3) = (10*12) + 2

G(3) =122

para x>1

se x=1, G(1)=1.

Desempilhando

- $G(x) = (10 * G(x-1)) + 2$

$$G(4) = (10*(G(3)) + 2$$
$$G(4) = (10*122) + 2$$
$$G(4) = 1222$$

para x>1
se x=1, G(1)=1.

x=4

(10*(G(3))+2 = 1222

Desempilhando

Definição

O que podemos concluir em ambos casos recursivos?

1. Entramos num processo repetitivo de chamar diversas vezes a mesma função;
2. A cada nova chamada, mudava-se o "variável de entrada";
3. Empilhávamos cada chamada;
4. O processo de chamadas finaliza quando alcançamos uma "variável de entrada" específica;
5. Desempilhávamos com os resultados obtidos;

Recursividade expressa por módulos/funções

- Nas aulas anteriores, vimos que um módulo pode invocar outro;
- Hoje, veremos **a situação em que uma função invoca a ela próprio**;
- Eles **não ocupam o mesmo espaço no memória**;
- Como nas variáveis locais vistas anteriormente, **as funções invocadas (variáveis) são empilhadas e desempilhadas**;

```
<tipo> funcao_recursiva (<tipo>var)
        se (var==caso_base)
              return resultado
        senão
              return funcao_recursiva (var-1)
```

Recursividade expressa por módulos/funções

Recursividade expressa por módulos/funções

Caso base

Passo recursivo

```
<tipo> funcao_recursiva (<tipo>var)
        se (var==caso_base)
              return resultado
        senão
              return funcao_recursiva (var-1)
```

Exemplo 1

$G(x) = 10 * G(x-1) + 2$, para $x > 1$. No caso de $x = 1$, $G(x) = 1$, neste caso:

- **Caso base:** $G(x) = 1$, se $x = 1$.
- **Passo recursivo:** $G(x) = 10 * G(x-1) + 2$

Recursividade expressa por módulos/funções

```
int rec(int n)
{
    if(n==1)
        return 1;
    else
        return 10*rec(n-1)+2;
}
```

```
int main( )
{ int x;
    printf("Digite um valor\n");
    scanf("%d",&x);
    printf("%d",rec(x));
}
```

Recursividade expressa por módulos/funções

Exemplo 2

$Fat(x) = x * Fat(x-1)$, para $x > 1$. No caso de $x = 1, Fat(x) = 1$, neste caso:

- **Caso base:** $Fat(x) = 1$, se $x = 1$.
- **Passo recursivo:** $Fat(x) = x * Fat(x-1)$

4! = 4 * 3 * 2 * 1 = 24
4 * 3!
3 * 2!
2 * 1!
1! = 1

x! = x * (x-1) * (x-2) * ... * 1 = resultado
x * (x-1)!

Hands on

```
int fat(int n)
{
    if(n==1)
        return 1;
    else
        return n*fat(n-1);
}
```

```
int main( )
{ int x;
    printf("Digite um valor\n");
    scanf("%d",&x);
    printf("%d",fat(x));
}
```

Façamos o teste de mesa

Hands on

Recursivo x iterativo

Vantagens

Código mais "enxuto" (conciso);
Mais fácil de ser compreendido;
Mais fácil de ser implementado em linguagens de programação;
Ótimo para definições matemáticas;

Desvantagens

Maior consumo de recursos;

Mais difíceis de serem testados se houverem muitas chamadas;


UÉ, POR QUE VOCÊ ESTÁ AÍ HÁ UM TEMPO OLHANDO PRA PAREDE?
ESTOU AQUI PENSANDO EM COMO POSSO RESOLVER UM PROBLEMA DE RECURSIVIDADE
Ask Google
MUITO DIFÍCIL O PROBLEMA?
UM TANTO...
Ask Google
Ask Google
JOGUE O LIXO NO LIXO
VIDA DE PROGRAMADOR .COM.BR
#1516
BY NC SA

Assistam aos vídeos

- Canal: Programação descomplicada linguagem C

  https://youtu.be/T2gTc5u-i1o (Aula 51: Recursividade)

  https://youtu.be/FH5lCr-RVWE (Aula 52: Recursão - Parte 2 - Funcionamento)

  https://youtu.be/o3MPTEc3LD8 (Aula 53 - Recursão – Cuidados)

  https://youtu.be/YEeYk9uEqEI (Aula 54 - Recursão – Soma até N)