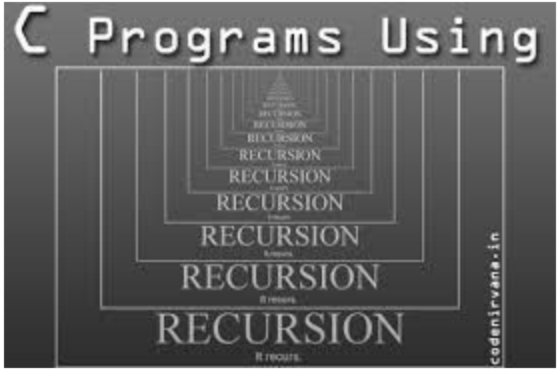
Recursividade

Prof. Denio Duarte
duarte@uffs.edu.br
Prof. Claunir Pavan
claunir.pavan@uffs.edu.br

Recursividade

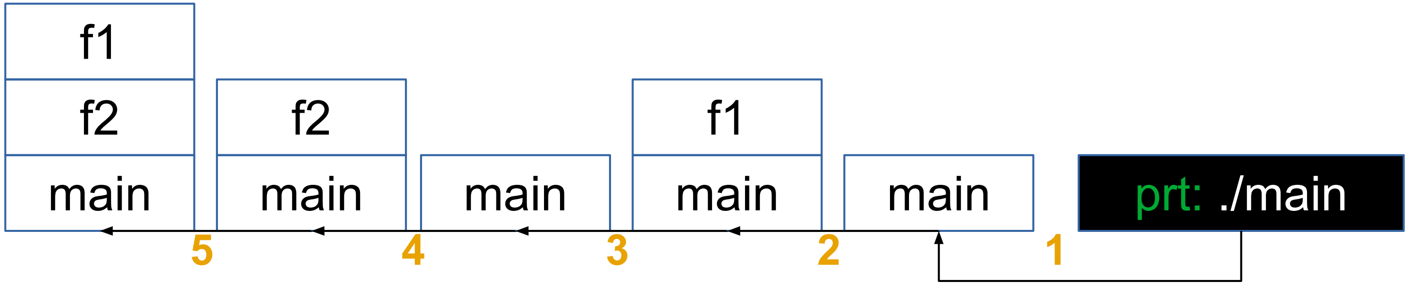
- É uma técnica de programação
- Baseada no conceito de uma função chamar uma instância dela mesma
- Alguns problemas são mais facilmente codificados utilizando a recursão



Recursividade

- Todo processo disparado por um programa ocupa um espaço da memória RAM
- Os processos de um programa são empilhados conforme a ordem que foram chamados (o último fica no topo da pilha)

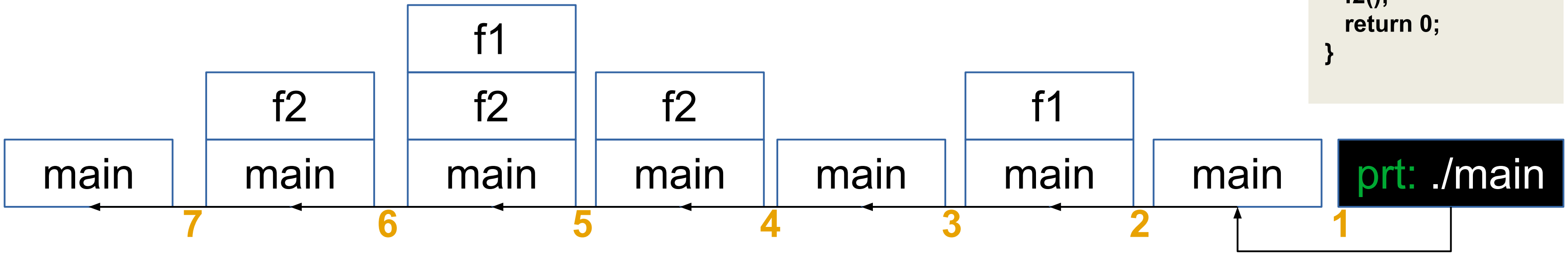
```
int f1(){
   print("Um\n");
   return 0;
}
int f2(){
   f1();
   return 1;
}
int main(){
   f1();
   f2();
   return 0;
}
```



Recursividade

- Todo processo disparado por um programa ocupa um espaço da memória RAM
- Os processos de um programa são empilhados conforme a ordem que foram chamados (o último fica no topo da pilha)

```
int f1(){
   print(“Um\n”);
   return 0;
}
int f2(){
   f1();
   return 1;
}
int main(){
   f1();
   f2();
   return 0;
}
```



Recursividade

- Funções iterativas
  - Funções tradicionais – que não se chamam
- Funções recursivas
  - São chamadas por elas mesmas
  - Podem causar um *looping* infinito

Recursividade

- Como usar isso para nosso benefício?
  - Quebramos o problema em partes menores, deixamos ele mais simples, e chamamos a função várias vezes até encontrar a resposta

Recursividade

- Como usar isso para nosso benefício?
  - Quebramos o problema em partes menores, deixamos ele mais simples, e chamamos a função várias vezes até encontrar a forma mais simples

Recursividade

- Como usar isso para nosso benefício?
  - Quebramos o problema em partes menores, deixamos ele mais simples, e chamamos a função várias vezes até encontrar a forma mais simples

- Podemos decompor uma recursão por
  - **Caso base**: uma instância do problema solucionada facilmente
  - **Chamadas recursivas**: onde a função é definida em termos de si própria, realizando uma redução para seu caso básico

Recursividade

- Exemplo 1:
  - Multiplicação através de adições
    - **3 x 4 = 3 + 3 + 3 + 3**, ou seja, 3+3+6 (3+3)=> 3 + 9 (3+6) => 12 (3+9)
  - Definição formal

$$\text{mult(m,n)} \begin{cases} \text{se n==0 então 0 (caso base)} \\ \text{caso contrário m+mult(m,n-1) (caso recursivo)} \end{cases}$$

Recursividade

mult(m,n) $\begin{cases} \text{se } n==0 \text{ então } 0 \text{ (caso base)} \\ \text{caso contrário } m+\text{mult}(m,n-1) \text{ (caso recursivo)} \end{cases}$

m m n

3 x 4 => 3 + (3 x 3)
3 + (3 x 2)
3 + (3 x 1)
3 + (3 x 0)

3 x 4 <= 3 + (9) 12
3 + (6) 9
3 + (3) 6
3 + (0) 3

Recursividade

mult(m,n) $\begin{cases} \text{se } n==0 \text{ então } 0 \text{ (caso base)} \\ \text{caso contrário } m+mult(m,n-1) \text{ (caso recursivo)} \end{cases}$

m m n

3 x 4 => 3 + (3 x 3)
3 + (3 x 2)
3 + (3 x 1)
3 + (3 x 0)

3 x 4 <= 3 + (9) 12
3 + (6) 9
3 + (3) 6
3 + (0) 3

```
int multrec (int m, int n)
{
  if (n==0) return 0;
  return m+multrec(m,n-1);
}
```

base

recursão

Recursividade

mult(m,n) { se n==0 então 0 (caso base)
caso contrário m+mult(m,n-1) (caso recursivo) }

	m		m	n
3 x 4 =>	3 +	(3 x	3)	
	3 +	(3 x	2)	
	3 +	(3 x	1)	

3 x 4 <= 3 + (9) 12
3 + (6) 9
3 + (3) 6
3 + (0) 3

```
int multite(int m, int n)
{
   int res=0,i;
   for (i=1;i<=n;i++)
      res+=m
   return res;
}
```

```
int multrec (int m, int n)
{
   if (n==0) return 0;
   return m+multrec(m,n-1);
}
```

base

recursão

Recursividade

- Exemplo 2
  - Fatorial
    - O fatorial de um número é o resultado da multiplicação do número por seus antecessores até 1 (por definição o fatorial de 0 é 1)
    - n!= n x (n-1) x (n-2) x … x 1
  - Definição formal

$$n! \begin{cases} \text{se } n{=}{=}0 \text{ ou } n{=}{=}1 \text{ então } 1 & \text{(caso base)} \\ \text{caso contrário } n \times (n-1)! & \text{(caso recursivo)} \end{cases}$$

Recursividade

n! $\begin{cases} \text{se } n==0 \text{ ou } n==1 \text{ então } 1 \text{ (caso base)} \\ \text{caso contrário } n \times (n-1)! \text{ (caso recursivo)} \end{cases}$

```
int fat(int n)
{
    if (n==0 || n==1)
      return 1;
    return n * fat(n-1);
}
```

base

recursão

Recursividade

n! {
se n==0 ou n==1 então 1 (caso base)
caso contrário n x (n-1)!   (caso recursivo)

```
int fat_it(int n)
{
   int r=1, i;
   if (n==0 || n==1)
      return r;
   for (i=1;i<=n;i++)
      r*=i;
   return r;
}
```

```
int fat(int n)
{
   if (n==0 || n==1)
      return 1;
   return n * fat(n-1);
}
```

base

recursão

Recursividade

- Exemplos “bobinhos”

```
void impvetasc(int *m, int t)
{
    if (t < 1) return;
    impvetasc(m,t-1);
    printf("%d\n",m[t-1]);
}
:
int v[4]={1,2,3,4};
impvetasc(v,4);
1
2
3
4
```

```
void impvetdesc(int *m, int t)
{
    if (t < 1) return;
    printf("%d\n",m[t-1]);
    impvetdesc(m,t-1);
}
:
int v[4]={1,2,3,4};
impvetdesc(v,4);
4
3
2
1
```

Exercícios

1. Implemente uma função recursiva que, dados dois números inteiros base (b) e expoente (e), calcula o valor de b^e ($e \geq 0$).

$$b^e \begin{cases} \text{se } e==0 \text{ então } 1 & \text{(caso base 1)} \\ \text{se } e==1 \text{ então } b & \text{(caso base 2)} \\ \text{caso contrário } b \times b^{(e-1)} & \text{(caso recursivo)} \end{cases}$$

2. Implemente uma função recursiva que calcule o somatório se um vetor passado por parâmetro.

$$\begin{cases} \text{se } n==0 \text{ então } v[0] & \text{(caso base 1)} \\ \text{caso contrário } v[n] + v[n-1] & \text{(caso recursivo)} \end{cases}$$

Exercícios

3. Implemente uma função recursiva que calcule o máximo divisor comum (mdc) entre dois números.

- Por exemplo, o mdc de 12 e 18 é 6

Definição (Algoritmo de Euclides):

$$\text{mdc}(m,n) \begin{cases} \text{se } n==0 \text{ então } m & \text{(caso base 1)} \\ \text{caso contrário } \text{mdc}(n, \underbrace{m\%n}_{\text{resto da divisão}}) & \text{(caso recursivo)} \end{cases}$$