Recursividade

Alguns problemas são definidos com base nos mesmos, ou seja, podem ser descritos por instâncias do próprio problema.

Para tratar estas classes de problemas, utiliza-se o conceito de recursividade.

Um módulo recursivo é um módulo que em sua seção de comandos chama a si mesmo.

Uma grande vantagem da recursividade é o fato de gerar uma redução no tamanho do algoritmo, permitindo descrevê-lo de forma mais clara e concisa.

UNIVASF

Recursividade

Porém, todo cuidado é pouco ao se fazer módulos recursivos. A primeira coisa a se providenciar é um critério de parada, o qual vai determinar quando o módulo deverá parar de chamar a si mesmo. Este cuidado impede que o módulo se chame infinitas vezes.

Um exemplo de um problema passível de definição recursiva é a operação de multiplicação efetuada sobre números naturais. Podemos definir a multiplicação em termos da operação mais simples de adição.

UNIVASF

Recursividade

No caso

$$A * B$$

pode ser definido como

$$A + A * (B - 1)$$

precisamos agora especificar um critério de parada. Qual seria?

$$A * 0 = 0$$

Recursividade

De acordo com o que vimos até o momento, podemos definir um laço de repetição que implementaria o cálculo da operação de multiplicação com base na operação de adição. Definiremos agora este laço:

```
...
A, B, RES: inteiro
...
RES<-0
enquanto (B<>0) faca
    RES <- RES + A
    B<-B-1
fimenquanto
...
```

Recursividade

Com base no que vimos podemos, também, definir um módulo recursivo que implemente a operação de multiplicação com base na operação de adição:

```
funcao multiplicar (A: inteiro; B: inteiro): inteiro
inicio
  se (B=0) entao
    retorne (0)
  senao
    retorne (A + multiplicar (A, B-1))
  fimse
fimfuncao
```

Recursividade

Para uma melhor compreensão do que foi apresentado, devemos compreender o conceito de “registro de ativação”.

O registro de ativação é uma área de memória que guarda informações referentes ao estado atual de um módulo ou do próprio algoritmo:

- valor dos parâmetros (para módulos);
- valor das variáveis locais (para módulos);
- valor do contador de programa (PC);
- etc.

UNIVASF

Recursividade

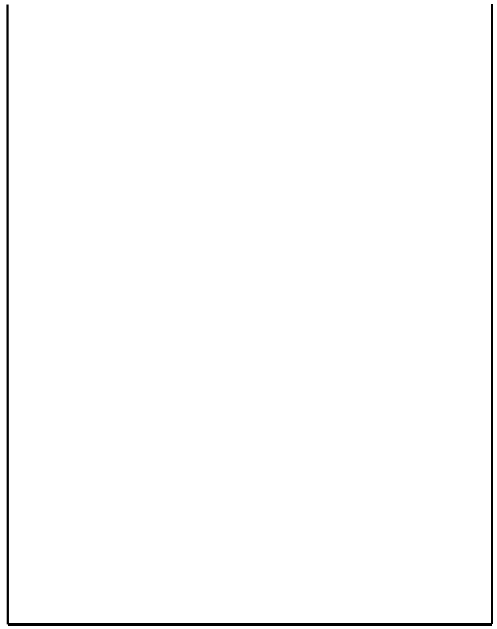
Sempre que um módulo é chamado o registro de ativação de quem o invocou (do algoritmo principal, de um outro módulo ou do próprio módulo) é salvo e um novo registro de ativação é criado para o módulo invocado. Estes registros de ativação são empilhados em uma pilha de registros de ativação.

Este processo é conhecido como salvamento e troca de contexto e pode ser melhor compreendido se o aplicarmos sobre um algoritmo que se utilize do módulo recursivo "multiplicar" definido anteriormente.

UNIVASF

```
algoritmo "exemplo recursividade"
var
   A, B, RES: inteiro
   funcao multiplicar (A: inteiro; B: inteiro): inteiro
   inicio
      1 se (B=0) entao
      2    retorne (0)
      3 senao
      4    retorne (A + multiplicar (A, B-1))
      5 fimse
   fimfuncao
inicio
 1 repita
 2    escreva ("Multiplicando (valor natural): ")
 3    leia (A)
 4 ate (A>=0)
 5 repita
 6    escreva ("Multiplicador (valor natural): ")
 7    leia (B)
 8 ate (B>=0)
 9 RES <- multiplicar(a,b)
10 escreva (a," *",b," =",res)
fimalgoritmo
```

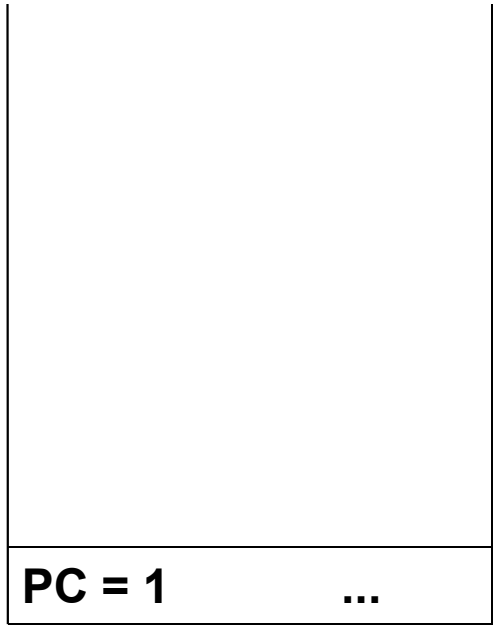
```
algoritmo "exemplo recursividade"
var
  A, B, RES: inteiro
  funcao multiplicar (A: inteiro; B: inteiro): inteiro
  inicio
     1 se (B=0) entao
     2   retorne (0)
     3 senao
     4   retorne (A + multiplicar (A, B-1))
     5 fimse
  fimfuncao
inicio
 1 repita
 2   escreva ("Multiplicando (valor natural): ")
 3   leia (A)
 4 ate (A>=0)
 5 repita
 6   escreva ("Multiplicador (valor natural): ")
 7   leia (B)
 8 ate (B>=0)
 9 RES <- multiplicar(a,b)
10 escreva (a," *",b," =",res)
fimalgoritmo
```



Pilha de registros de ativação

Para melhor contextualizar nossa explicação vamos presumir que o usuário forneceu o valor 7 para “A” e o valor 3 para “B”.

```
algoritmo "exemplo recursividade"
var
  A, B, RES: inteiro
  funcao multiplicar (A: inteiro; B: inteiro): inteiro
  inicio
     1 se (B=0) entao
     2   retorne (0)
     3 senao
     4   retorne (A + multiplicar (A, B-1))
     5 fimse
  fimfuncao
inicio
 1 repita
 2   escreva ("Multiplicando (valor natural): ")
 3   leia (A)
 4 ate (A>=0)
 5 repita
 6   escreva ("Multiplicador (valor natural): ")
 7   leia (B)
 8 ate (B>=0)
 9 RES <- multiplicar(a,b)
10 escreva (a," *",b," =",res)
fimalgoritmo
```

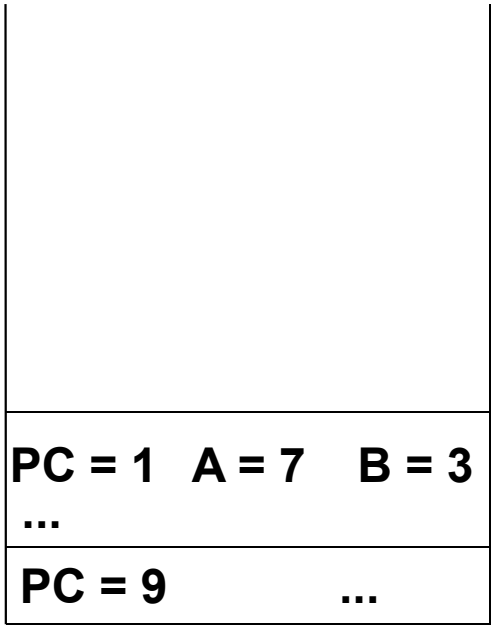


Pilha de registros de ativação

Inicialmente o registro de ativação do algoritmo é colocado na pilha de registros de ativação.

```
algoritmo "exemplo recursividade"
var
  A, B, RES: inteiro
  funcao multiplicar (A: inteiro; B: inteiro): inteiro
  inicio
     1 se (B=0) entao
     2   retorne (0)
     3 senao
     4   retorne (A + multiplicar (A, B-1))
     5 fimse
  fimfuncao
inicio
 1 repita
 2   escreva ("Multiplicando (valor natural): ")
 3   leia (A)
 4 ate (A>=0)
 5 repita
 6   escreva ("Multiplicador (valor natural): ")
 7   leia (B)
 8 ate (B>=0)
 9 RES <- multiplicar(a,b)
10 escreva (a," *",b," =",res)
fimalgoritmo
```

RES <- multiplicar (7,3)
?

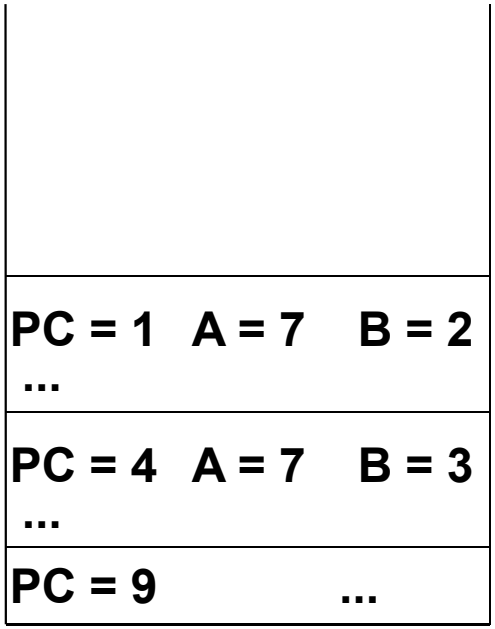


Pilha de registros de ativação

Em nosso exemplo a primeira execução de um módulo ocorre na nona instrução da seção de comandos do algoritmo. Neste momento é salvo o registro de ativação do algoritmo e introduzido na pilha um novo registro de ativação referente ao módulo chamado.

```
algoritmo "exemplo recursividade"
var
  A, B, RES: inteiro
  funcao multiplicar (A: inteiro; B: inteiro): inteiro
  inicio
     1 se (B=0) entao
     2   retorne (0)
     3 senao
     4   retorne (A + multiplicar (A, B-1))
     5 fimse
  fimfuncao
inicio
 1 repita
 2   escreva ("Multiplicando (valor natural): ")
 3   leia (A)
 4 ate (A>=0)
 5 repita
 6   escreva ("Multiplicador (valor natural): ")
 7   leia (B)
 8 ate (B>=0)
 9 RES <- multiplicar(a,b)
10 escreva (a," *",b," =",res)
fimalgoritmo
```

RES <- multiplicar (7,3)
?
7 + multiplicar (7,2)
?



Pilha de registros de ativação

Devido ao valor contido no parâmetro B a quarta instrução da seção de comandos do módulo é executada chamando novamente o módulo multiplicar. Neste momento é salvo um registro de ativação (RA) do invocador e introduzido na pilha um novo RA.

```
algoritmo "exemplo recursividade"
var
  A, B, RES: inteiro
  funcao multiplicar (A: inteiro; B: inteiro): inteiro
  inicio
     1 se (B=0) entao
     2   retorne (0)
     3 senao
     4   retorne (A + multiplicar (A, B-1))
     5 fimse
  fimfuncao
inicio
 1 repita
 2   escreva ("Multiplicando (valor natural): ")
 3   leia (A)
 4 ate (A>=0)
 5 repita
 6   escreva ("Multiplicador (valor natural): ")
 7   leia (B)
 8 ate (B>=0)
 9 RES <- multiplicar(a,b)
10 escreva (a," *",b," =",res)
fimalgoritmo
```

RES <- multiplicar (7,3)
?
7 + multiplicar (7,2)
?
7 + multiplicar (7,1)
?

PC = 1 A = 7 B = 1 ...
PC = 4 A = 7 B = 2 ...
PC = 4 A = 7 B = 3 ...
PC = 9 ...

Pilha de registros de ativação

Devido ao valor contido no parâmetro B a quarta instrução da seção de comandos do módulo é executada chamando novamente o módulo multiplicar. Neste momento é salvo o RA do invocador e introduzido na pilha um novo RA.

Recursividade

Devido ao valor contido no parâmetro B a quarta instrução da seção de comandos do módulo é executada chamando novamente o módulo multiplicar. Neste momento é salvo o RA do invocador e introduzido na pilha um novo RA.

Devido ao valor contido no parâmetro B a segunda instrução da seção de comandos do módulo é executada retornando o valor zero e finalizando as chamadas recursivas. Neste momento é salvo o RA do invocador e introduzido na pilha um novo RA.

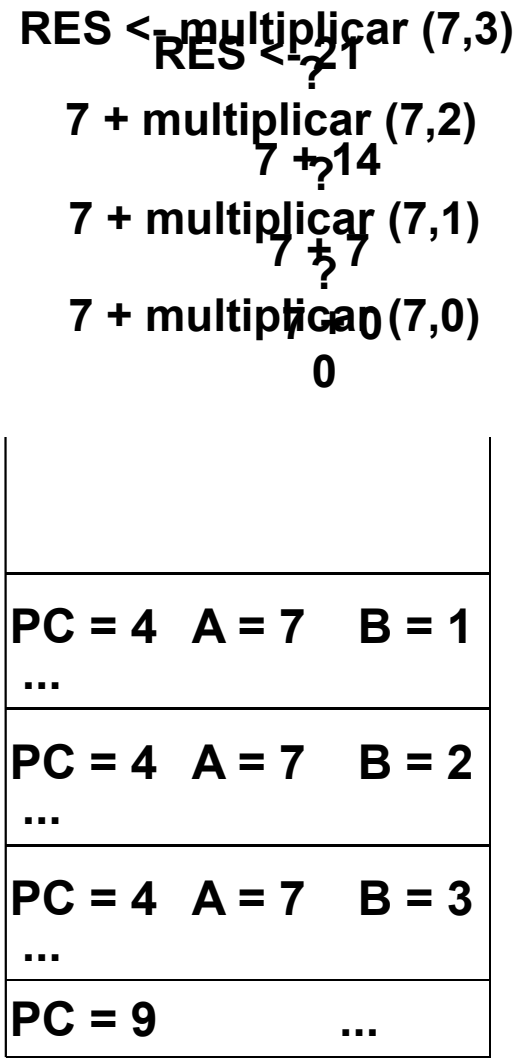
RES <- multiplicar (7,3)
?
7 + multiplicar (7,2)
?
7 + multiplicar (7,1)
?
7 + multiplicar (7,0)
0

PC = 1	A = 7	B = 0
...		
PC = 4	A = 7	B = 1
...		
PC = 4	A = 7	B = 2
...		
PC = 4	A = 7	B = 3
...		
PC = 9		...

Pilha de registros de ativação

Recursividade

Desta forma um a um os módulos vão sendo finalizados e seus registros de ativação desempilhados.



Pilha de registros de ativação

Recursividade

Com base no que foi exposto, podemos visualizar algumas desvantagens da utilização de recursividade, como:

- O consumo de memória necessário para a troca de contexto.

- Redução do desempenho de execução devido ao tempo para gerenciamento de chamadas.

- Dificuldades na depuração de programas recursivos, especialmente se a recursão for muito profunda.

Exercício 47: Para uma melhor compreensão do conceito de recursividade faça agora um módulo recursivo para calcular o fatorial de um número natural e construa um algoritmo que o utilize de forma adequada do módulo em questão.

UNIVASF

```
algoritmo "fatorial recursivo"
var
   n: inteiro
   funcao fatorial (num: inteiro): inteiro
   inicio
      se (num=0) entao
         retorne (1)
      senao
         retorne (num * fatorial(num-1))
      fimse
   fimfuncao
inicio
   escreva("Digite o número que você deseja saber o fatorial: ")
   leia (n)
   se (n>=0) entao
      escreva ("O fatorial do número ",n," é ",fatorial(n))
   senao
      escreva("Não existe fatorial de números negativos!")
   fimse
fimalgoritmo
```

Recursividade

Um outro exemplo muito utilizado de problema que possui uma definição recursiva é a geração da série de Fibonacci:

{0,1,1,2,3,5,8,13,21,34, …}

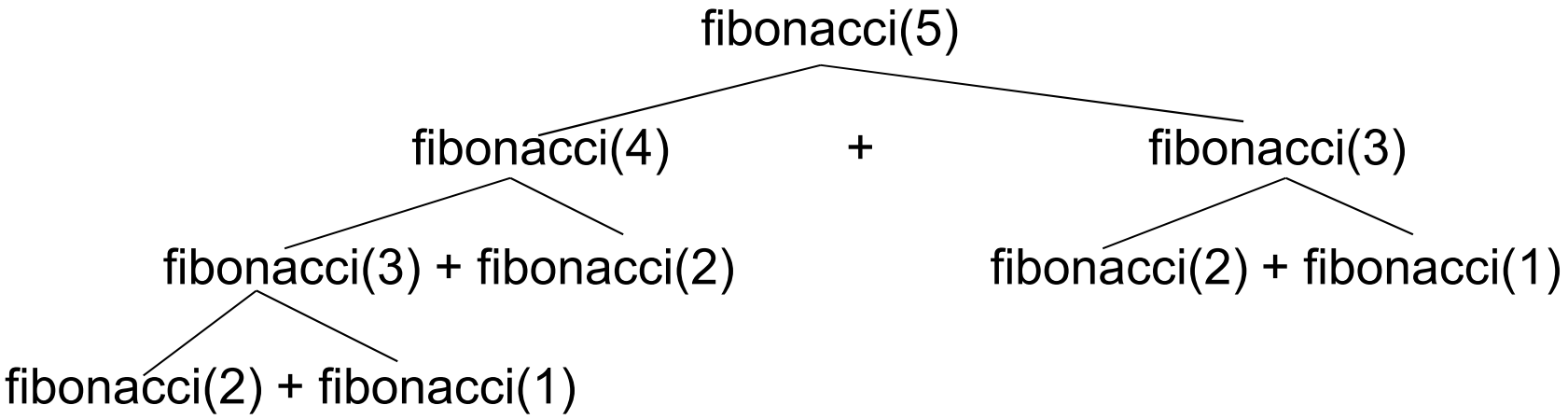
Uma função recursiva que recebe a posição do elemento na série e retorna seu valor é:

```
funcao fibonacci (i: inteiro): inteiro
inicio
   se (i=1) entao
      retorne (0)
   senao
      se (i=2) entao
         retorne (1)
      senao
          retorne (fibonacci(i-1) + fibonacci(i-2))
      fimse
   fimse
fimfuncao
```

Recursividade

Fora os problemas mencionados, gerados pela recursão, qual seria outro problema proveniente da recursão evidenciado na função recursiva apresentada para o cálculo do valor de um elemento da série de Fibonacci com base na sua posição?

O cálculo do mesmo elemento da séria n vezes.



Obs.: Mesmo problemas que possuem uma definição recursiva também podem ser solucionados de forma imperativa. Um exemplo disso é o cálculo do valor de um elemento da série de Fibonacci com base na sua posição através da função imperativa abaixo:

```
funcao fibonacci (i: inteiro): inteiro
var a, b:inteiro
inicio
   se (i=1) entao
      retorne (0)
   senao
      se (i=2) entao
         retorne (1)
      senao
         a<-0
         b<-1
         enquanto (i-2<>0) faca
            b<-b+a
            a<-b-a
            i<-i-1
         fimenquanto
         retorne (b)
      fimse
   fimse
fimfuncao
```

Recursividade

Assim como a série de Fibonacci existem outras sequências definidas por recorrência, ou seja, onde um valor da sequência é definido em termos de um ou mais valores anteriores, o que é denominado de relação de recorrência.

Exercício 48:

Estabeleça a relação de recorrência presente na sequência abaixo e construa uma função recursiva que receba a posição do elemento na série e retorne seu valor.

S = { 2, 4, 8, 16, 32 ... }

UNIVASF

Recursividade

A sequência S é definida por recorrência por

1. S(1) = 2
2. S(n) = 2 * S(n-1) para n>=2

Uma função recursiva que recebe a posição do elemento na série e retorna seu valor é:

```
funcao func (p: inteiro): inteiro
inicio
   se (p=1) entao
      retorne (2)
   senao
      retorne (2*func(p-1))
   fimse
fimfuncao
```

Exercício 49:

Elabore um módulo recursivo que receba dois números inteiros, como parâmetros, e retorne o resultado do somatório de todos os números contidos no intervalo aberto delimitado pelos números fornecidos. Em seguida, construa um algoritmo que se utilize de forma eficaz da módulo elaborado.