Programacao em computadores II

Recursividade

Prof. Dr. Fábio Rodrigues de la Rocha

Definição:

Recursividade é um termo usado de maneira geral para descrever o processo de repetição de um objeto de um jeito similar ao que já fora mostrado; Um bom exemplo disso são as imagens repetidas que aparecem quando dois espelhos são apontados um para o outro;

- Na matemática e na ciência da computação, a recursão especifica uma classe de objetos ou métodos definindo alguns poucos casos base ou métodos muito simples, e a partir disto definem-se regras para formular casos complexos.
- Exemplo de definição recursiva da ancestralidade de uma pessoa:
  - Os pais de uma pessoa são seus antepassados (caso base);
  - Os pais de qualquer antepassado são também antepassados da pessoa em consideração (passo recursivo).

Algoritmos recursivos

- Um método comum de simplificação consiste em dividir um problema em subproblemas do mesmo tipo.
- Como técnica de programação, isto se denomina divisão e conquista, e constitui a chave para o desenvolvimento de muitos algoritmos importantes, por exemplo, algoritmos de ordenação e busca.
- Praticamente todas as linguagens de programação atuais possibilitam o uso de funções e procedimentos recursivos.
- Toda função que puder ser resolvida por computador pode ser elaborada como uma função recursiva sem o uso de iterações.
- Do mesmo modo qualquer função recursiva pode ser descrita através de sucessivas iterações.

Algoritmos recursivos

- Em geral, uma definição recursiva é especificada através de um número limitado de casos base e um caso recursivo.
- Por exemplo, em uma função para calcular o N primeiros números positivos o caso base é N=1. No caso recursivo, dado N $>$ 1 o valor final é calculado pela soma de N + (N-1) até que N atinja 1;
- Outro exemplo, é o calculo do fatorial. No caso base o valor de 0! é 1. No caso recursivo, dado um N $>$ 0, o valor de N! é calculado multiplicando por N o valor de (N-1)!, e assim por diante, de tal forma que N! tem como valor N * (N-1) * (N-2) * ... * (N-N)!, onde (N-N)! representa obviamente o caso base.

Algoritmos recursivos

- No exemplo do fatorial, a implementação iterativa tende a ser ligeiramente mais rápida na prática do que a implementação recursiva.
- Isto acontece pois uma implementação recursiva precisa registrar o estado atual do processamento de maneira que ela possa continuar de onde parou após a conclusão de cada nova execução subordinada do procedimento recursivo. Esta ação consome tempo e memória.
- Existem outros tipos de problemas cujas soluções são inerentemente recursivas, já que elas precisam manter registros de estados anteriores.
- Exemplos: percurso de uma árvore; algoritmos de divisão e conquista, tais como o Quicksort.

Recursão - introdução

Analise o código abaixo:

```
1  #include <stdio.h>
2  #include <string.h>
3  int Fatorial (int x) {
4    int k, s=1;
5    for (k=0;k<x;k++)
6    {
7      s=s*(k+1);
8    }
9    return s;
10 }
11 int main (void) {
12    printf("Fat=%d\n",Fatorial(5));
13 }
```

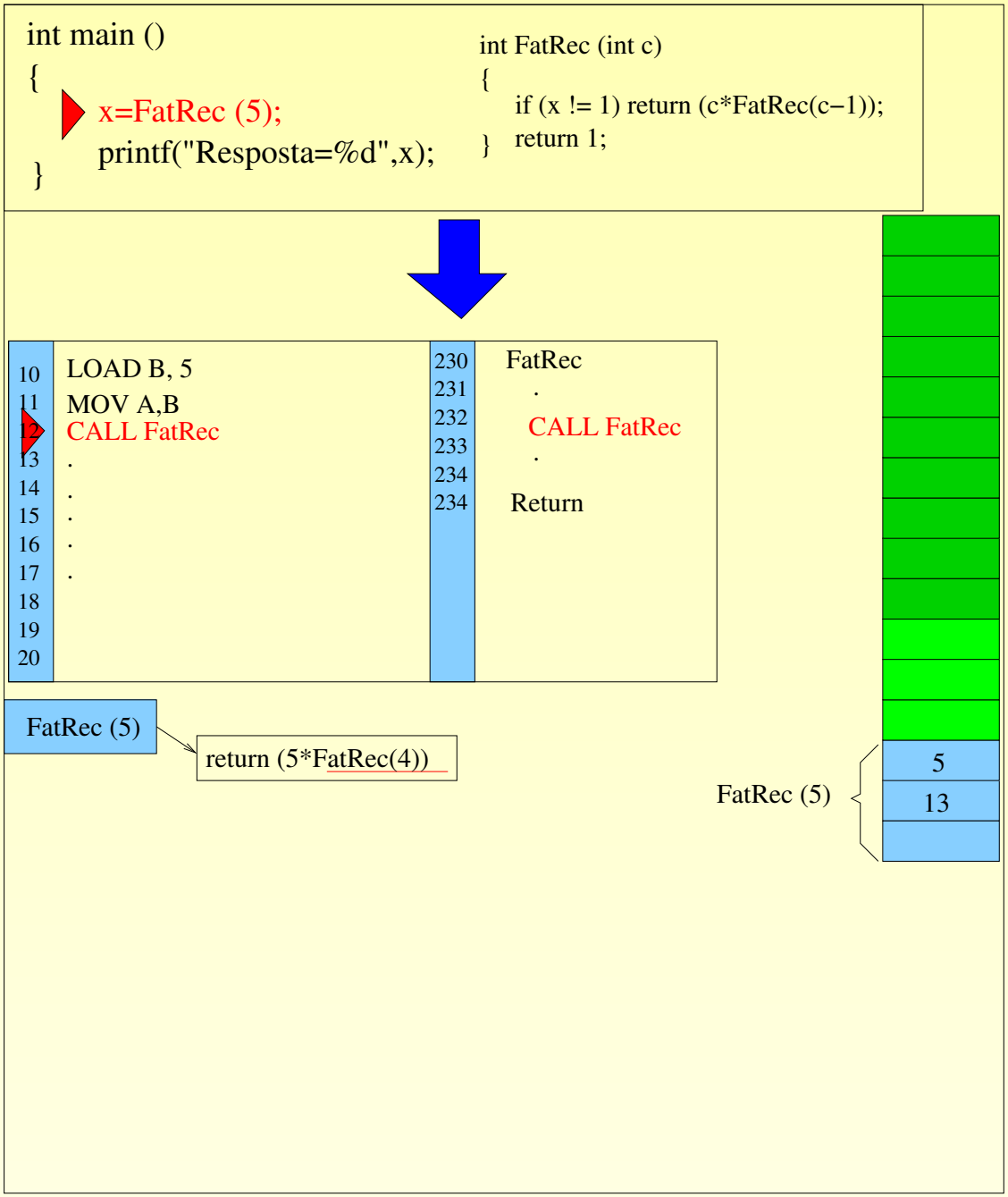
Recursão - introdução

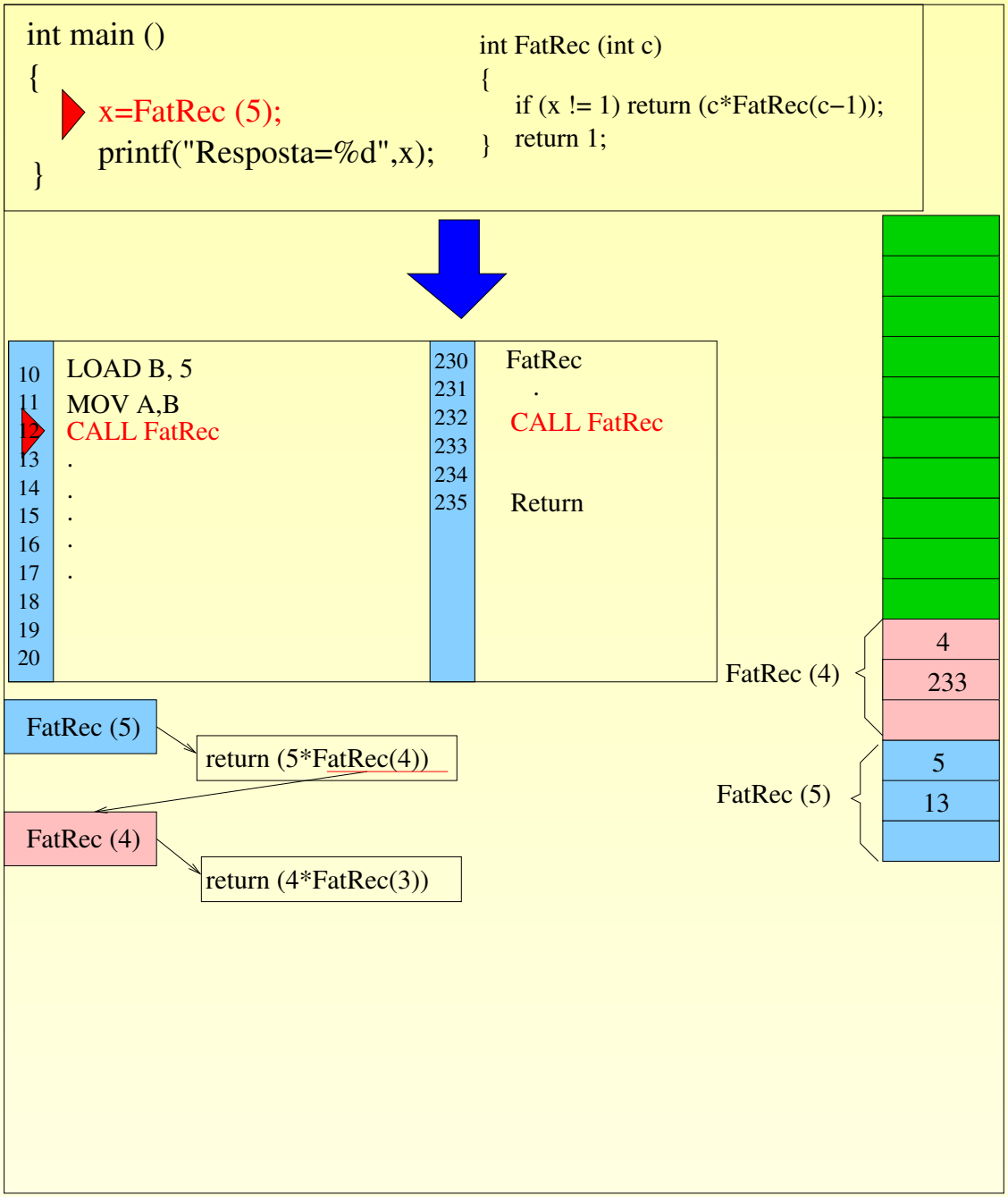
Analise o código abaixo:

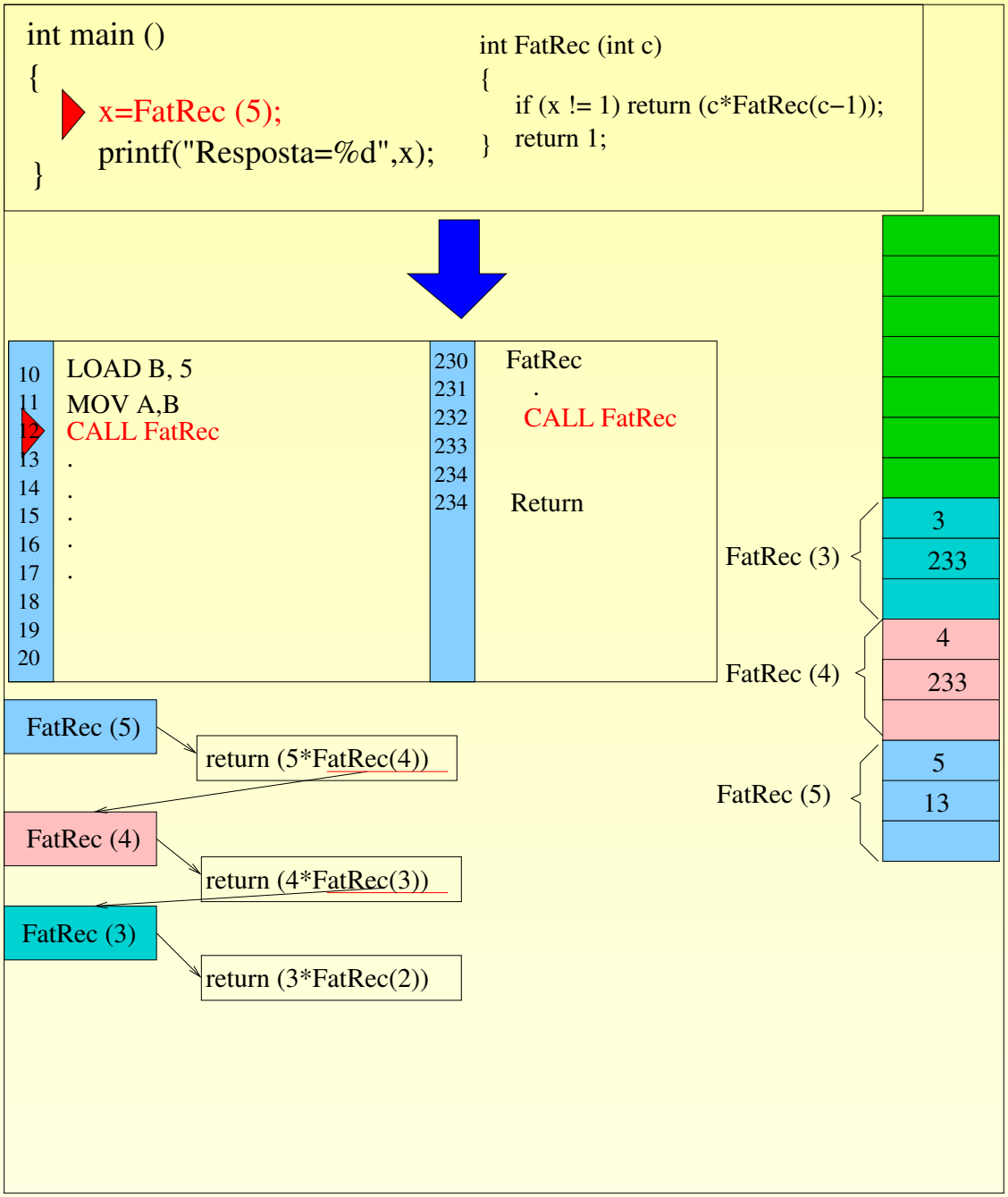
```
1  #include <stdio.h>
2  #include <string.h>
3
4  int FatRec( int x)
5  {
6  if (x!=1) return x*FatRec(x-1);
7  return (1); // no precisa de else
8  }
9
10 int main (void)
11 {
12    printf("FatRec=%d\n",FatRec(5));
13 }
```

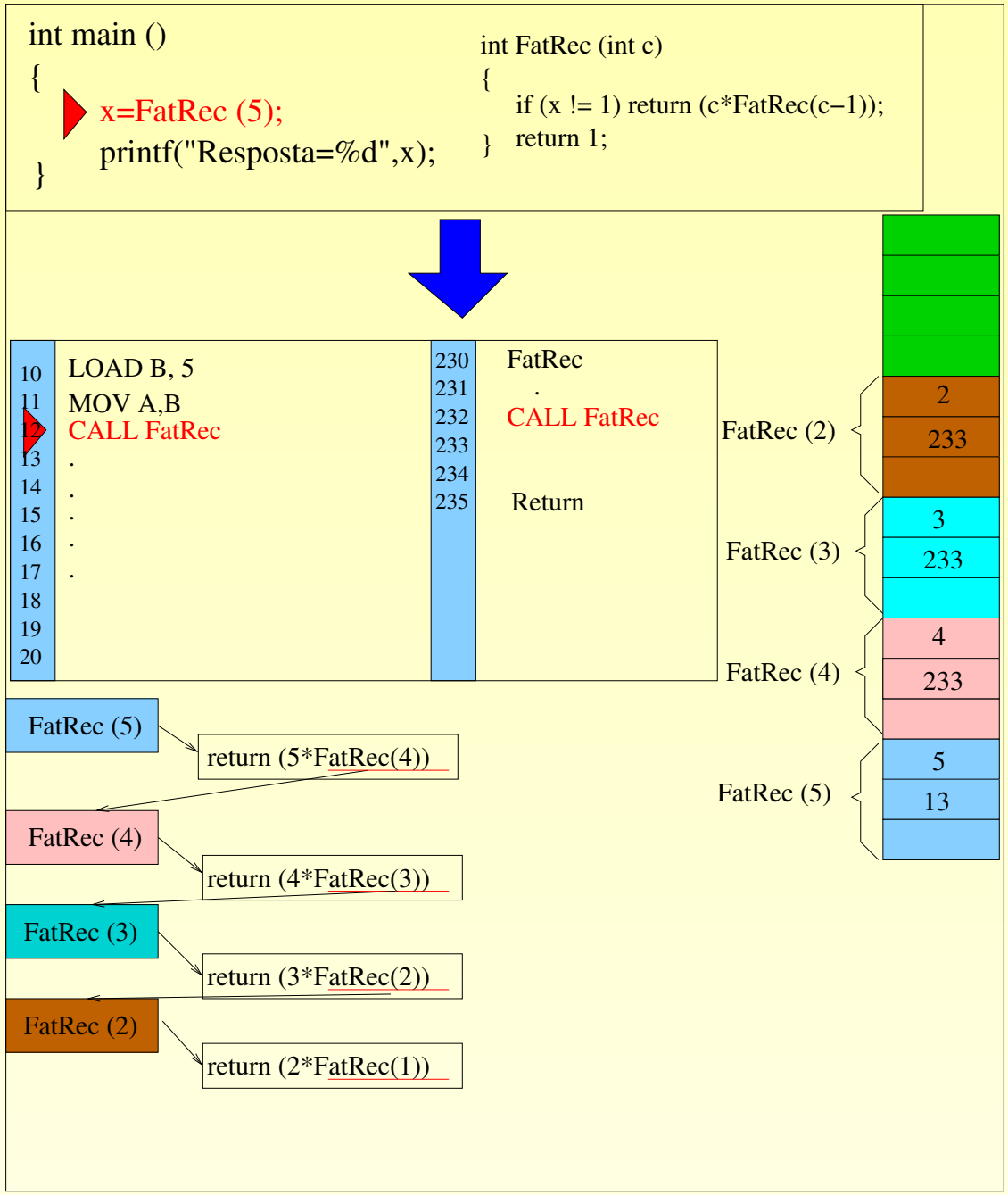
Recursão

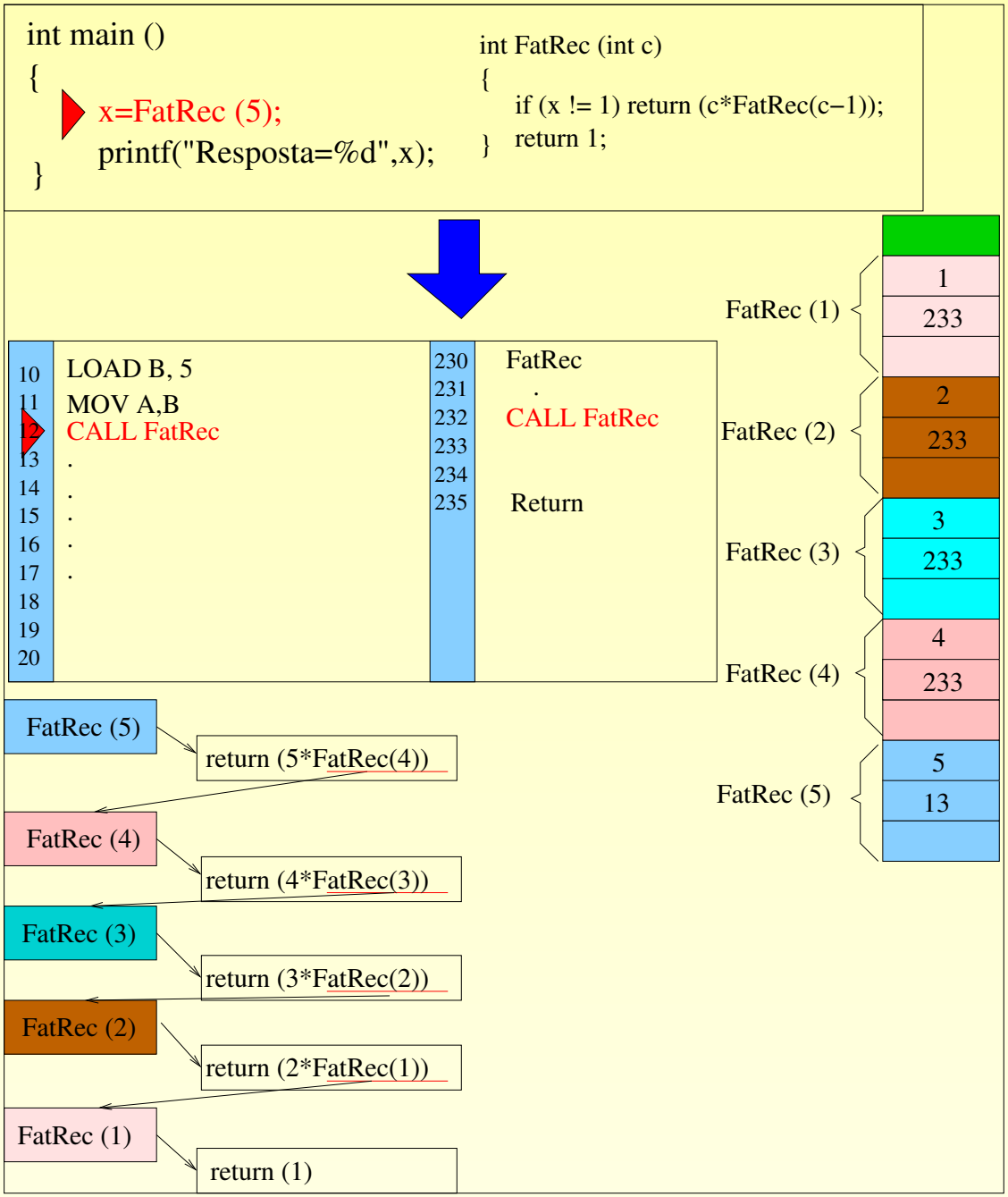
Como funciona internamente ?

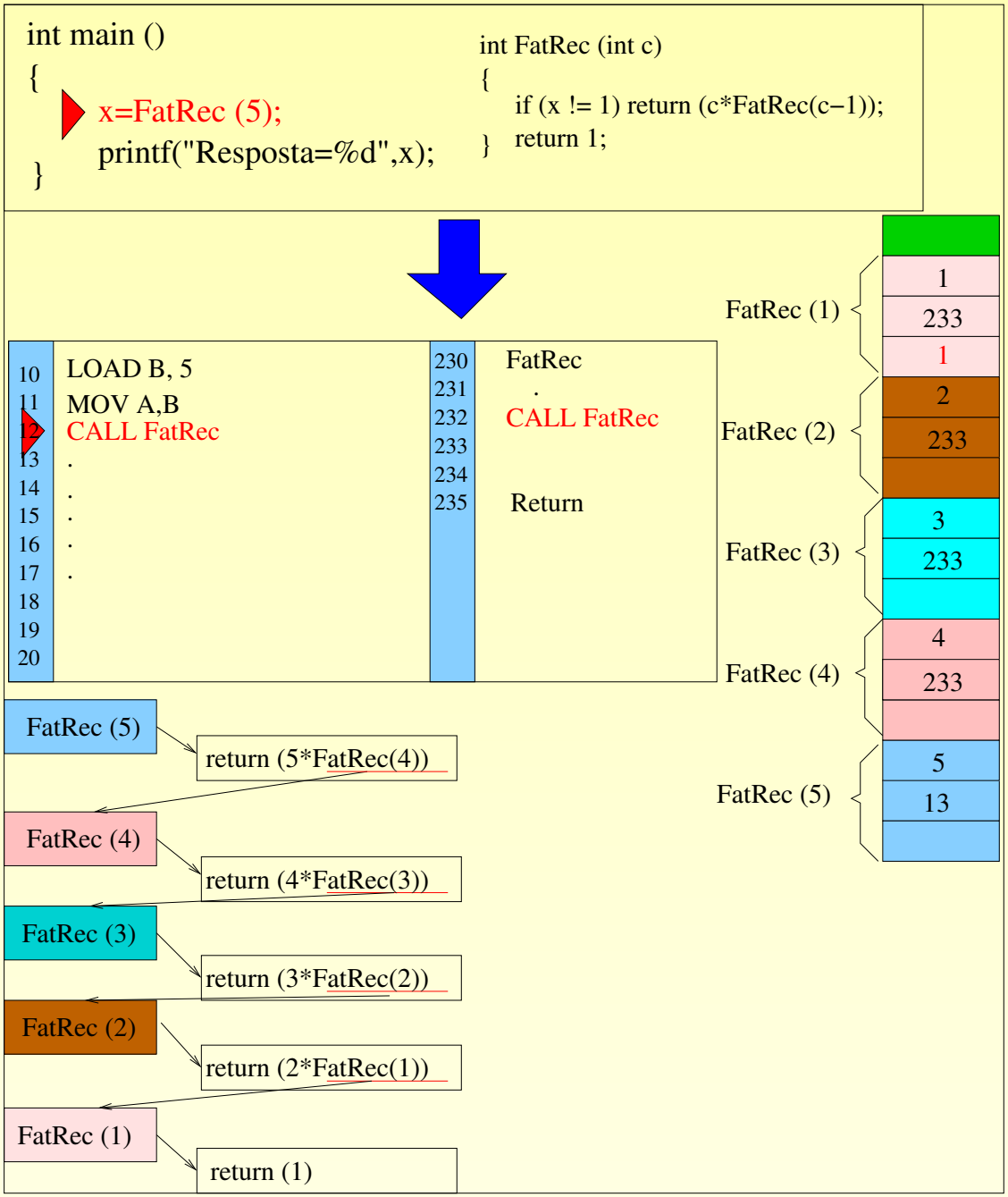

int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
234
FatRec
.
CALL FatRec
.
Return
FatRec (5)
return (5*FatRec(4))
5
13
FatRec (5)

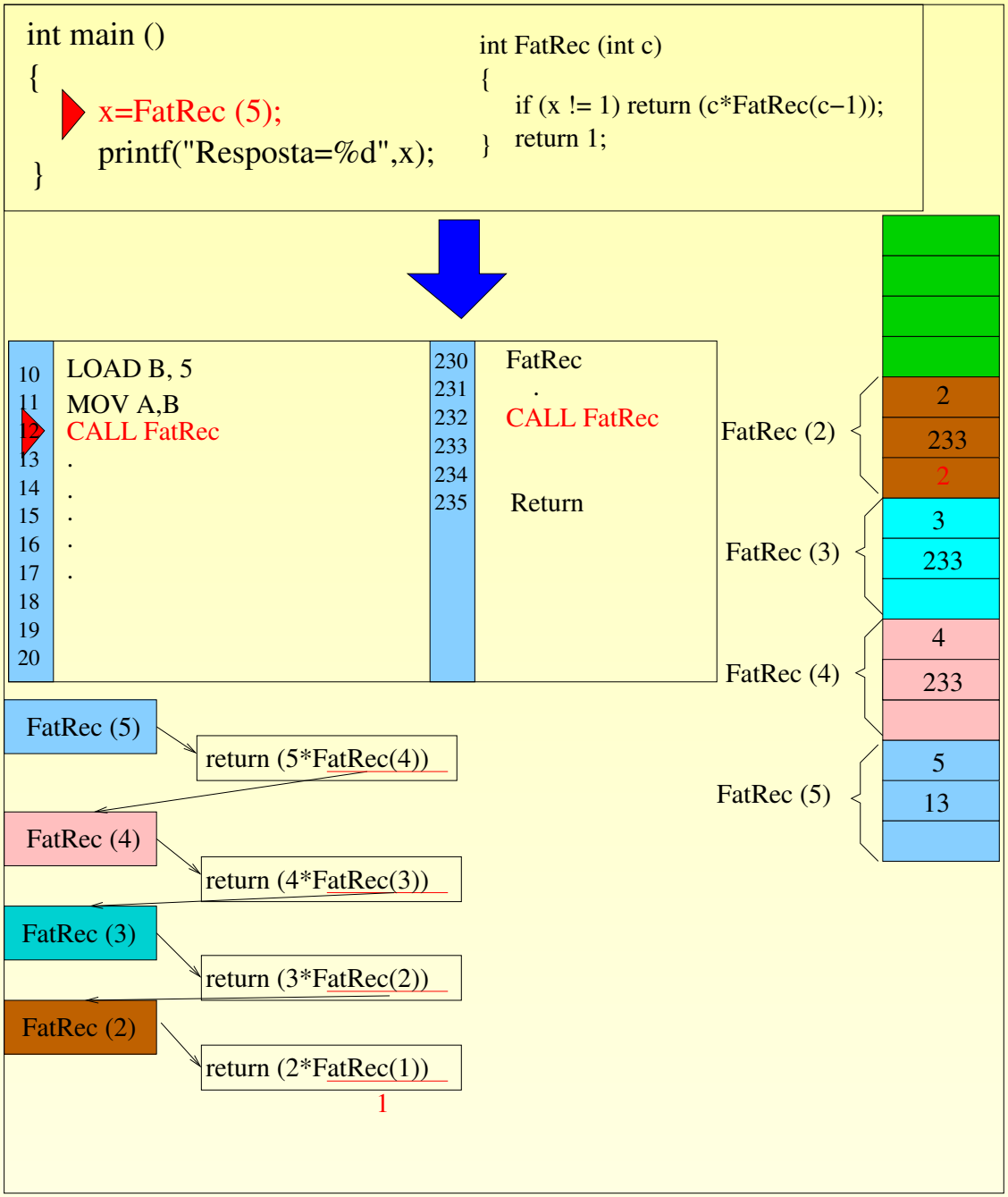

int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
235
FatRec
.
CALL FatRec
Return
4
233
FatRec (4)
5
13
FatRec (5)
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))

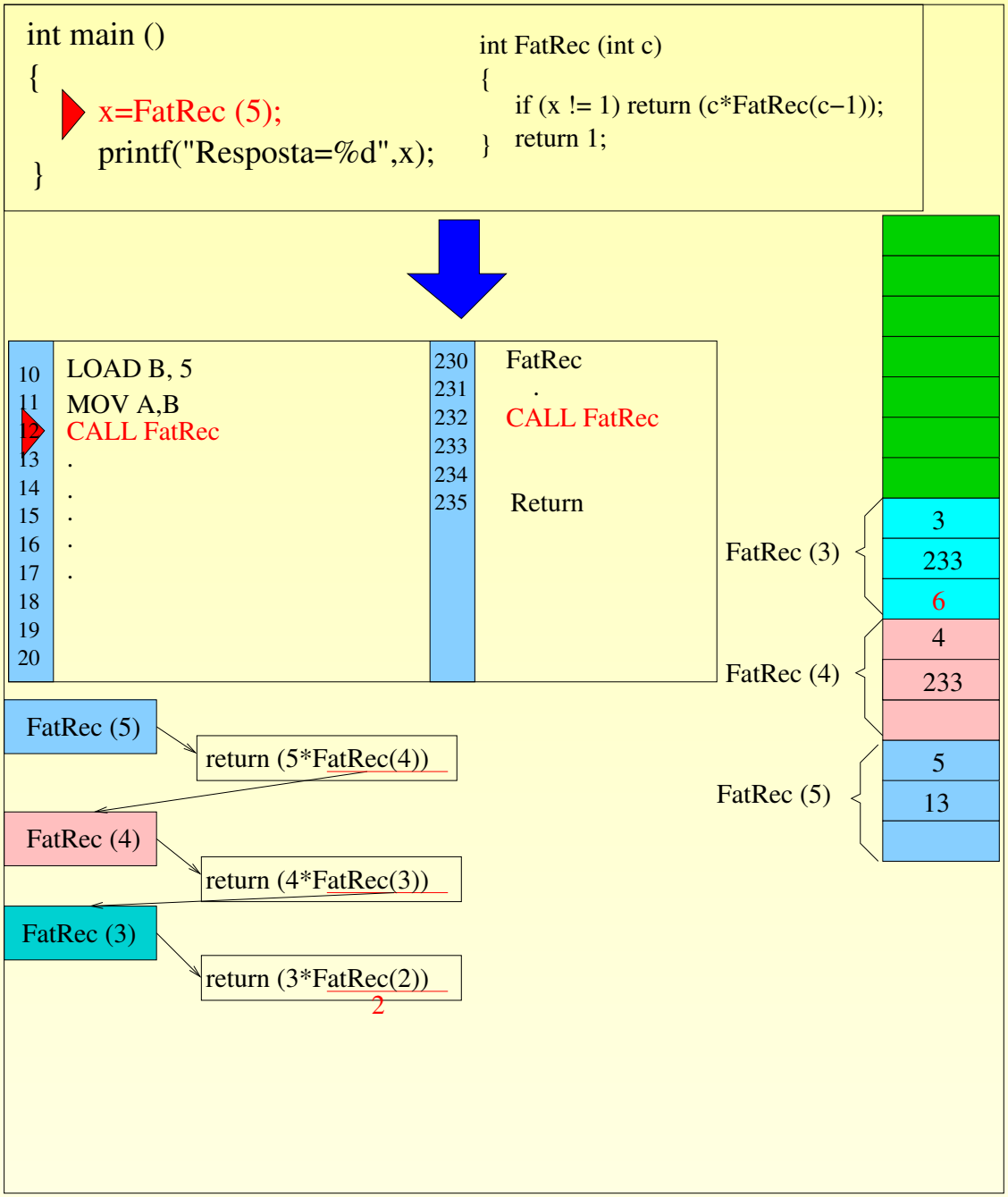

int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
234
FatRec
.
CALL FatRec
Return
3
233
4
233
5
13
FatRec (3)
FatRec (4)
FatRec (5)
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
FatRec (3)
return (3*FatRec(2))

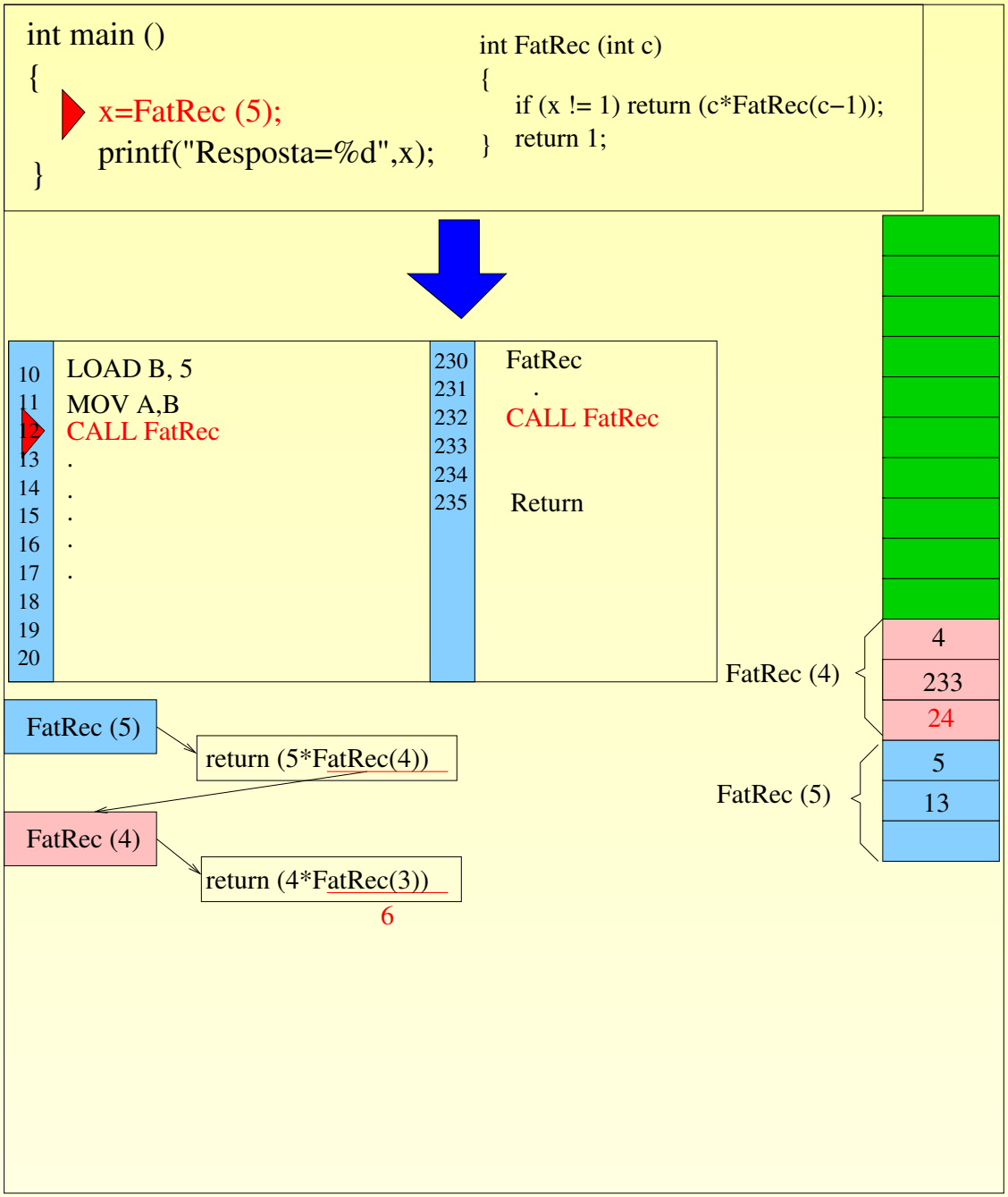

int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10 LOAD B, 5
11 MOV A,B
12 CALL FatRec
13 .
14 .
15 .
16 .
17 .
18
19
20
230 FatRec
231 .
232 CALL FatRec
233
234
235 Return
FatRec (2)
2
233
FatRec (3)
3
233
FatRec (4)
4
233
FatRec (5)
5
13
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
FatRec (3)
return (3*FatRec(2))
FatRec (2)
return (2*FatRec(1))

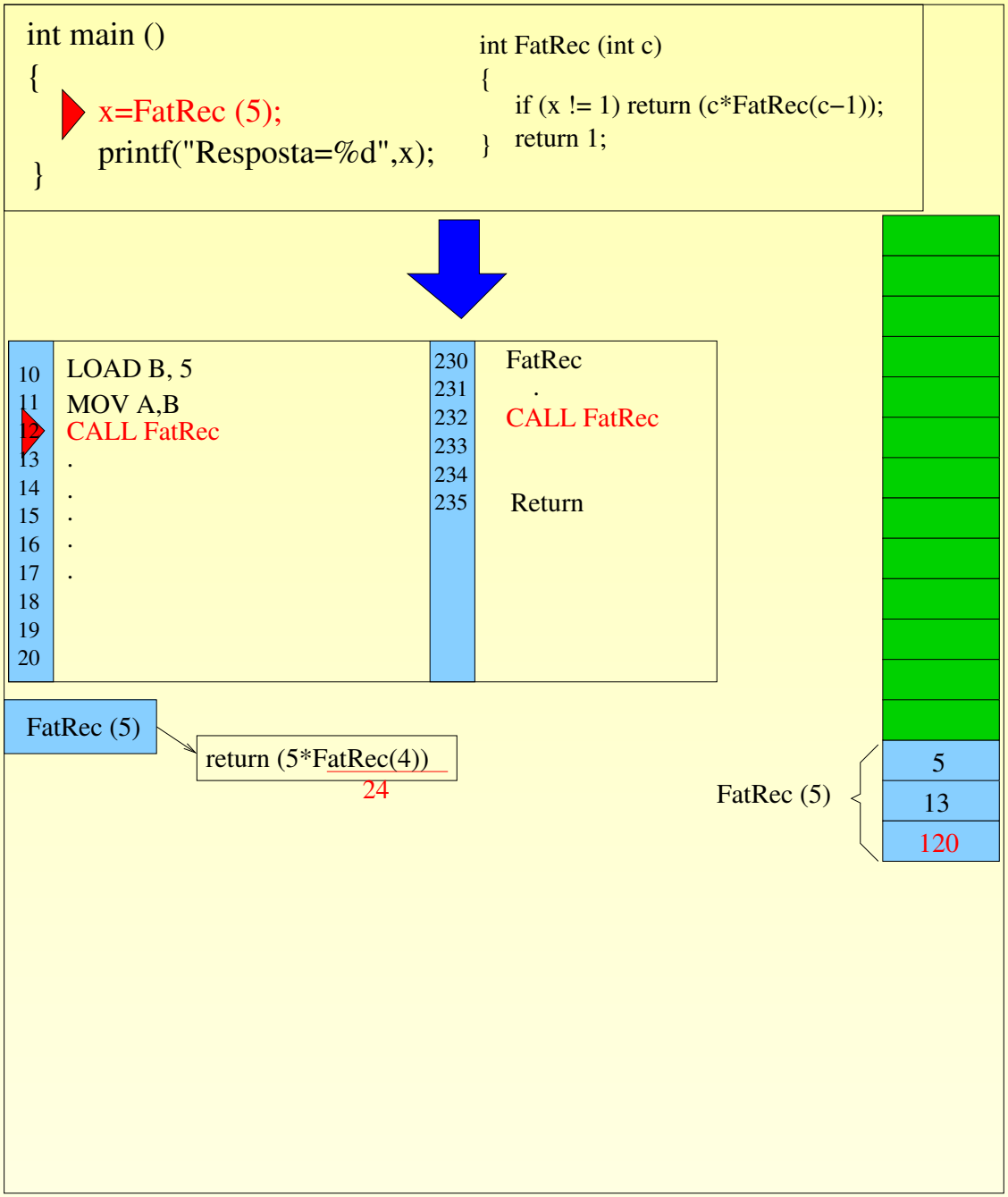

int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10 LOAD B, 5
11 MOV A,B
12 CALL FatRec
13 .
14 .
15 .
16 .
17 .
18
19
20
230 FatRec
231 .
232 CALL FatRec
233
234
235 Return
FatRec (1)
1
233
FatRec (2)
2
233
FatRec (3)
3
233
FatRec (4)
4
233
FatRec (5)
5
13
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
FatRec (3)
return (3*FatRec(2))
FatRec (2)
return (2*FatRec(1))
FatRec (1)
return (1)


int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10 LOAD B, 5
11 MOV A,B
12 CALL FatRec
13 .
14 .
15 .
16 .
17 .
18
19
20
230 FatRec
231 .
232 CALL FatRec
233
234
235 Return
FatRec (1)
1
233
1
FatRec (2)
2
233
FatRec (3)
3
233
FatRec (4)
4
233
FatRec (5)
5
13
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
FatRec (3)
return (3*FatRec(2))
FatRec (2)
return (2*FatRec(1))
FatRec (1)
return (1)


int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
235
FatRec
.
CALL FatRec
Return
FatRec (2)
FatRec (3)
FatRec (4)
FatRec (5)
2
233
2
3
233
4
233
5
13
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
FatRec (3)
return (3*FatRec(2))
FatRec (2)
return (2*FatRec(1))
1


int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
235
FatRec
.
CALL FatRec
Return
3
233
6
4
233
5
13
FatRec (3)
FatRec (4)
FatRec (5)
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
FatRec (3)
return (3*FatRec(2))
2


int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
235
FatRec
.
CALL FatRec
Return
4
233
24
5
13
FatRec (4)
FatRec (5)
FatRec (5)
return (5*FatRec(4))
FatRec (4)
return (4*FatRec(3))
6


int main ()
{
x=FatRec (5);
printf("Resposta=%d",x);
}
int FatRec (int c)
{
if (x != 1) return (c*FatRec(c−1));
return 1;
}
10
11
12
13
14
15
16
17
18
19
20
LOAD B, 5
MOV A,B
CALL FatRec
.
.
.
.
.
230
231
232
233
234
235
FatRec
.
CALL FatRec
Return
FatRec (5)
return (5*FatRec(4))
24
5
13
120
FatRec (5)

Outro exemplo de recursão

Somatório dos N primeiros números

Digamos que seja necessário criar uma função que retorne a soma dos N primeiros números inteiros. Ou seja, se N=3 a soma será 6 visto que 1+2+3=6

Este algoritmo pode ser criado de forma **iterativa**:

```
1 int soma (int N)
2 {
3    int x,S=0;
4
5    for (x=1;x<=N;x++) S=S+x;
6    return S;
7 }
```

Outro exemplo de recursão - continuação

Somatório dos N primeiros números

O algoritmo pode ser escrito na forma **recursiva** se percebermos que $soma(3) = 3 + soma(2)$

```
1 int soma (int N)
2 {
3    if (N==1) return 1;
4    return N+soma(N-1);
5 }
```

Mais um exemplo de recursão

Imprimir um número inteiro dígito por dígito usando recursão

Dica: escolha um número **n** e veja como o algoritmo se comporta

```
1 void printd (int n) {
2    if (n < 0) { /* imprime sinal */
3       putchar('-');
4       n = -n;
5    }
6    if ((n / 10) != 0) printd(n/10); /* recursao se n>10 */
7    putchar(n % 10 + '0');
```

Mais um exemplo de recursão

Sequência de Fibonacci

A sequência de fibonacci é uma sequência tal que o primeiro termo é 0, o segundo termo é 1 e os demais termos são dados pela soma dos dois últimos números.
$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$ ou ainda:
$fibonacci(0) = 0;$
$fibonacci(1) = 1;$
$fibonacci(n) = fibonacci(n-1) + fibonacci(n-2);$

```
1 long fibonacci(long n) {
2    if ( n == 0 || n == 1) return n;
3    return (fibonacci (n-1) + fibonacci(n-2));
4 }
```

Mais um exemplo de recursão

Converter um número inteiro em string

```
1  void itoa_aux(int num, char * &s) {
2     if ( num == 0 ) return ;
3     itoa_aux( int(num / 10), s);
4     *s++ = n % 10 + '0';
5  }
6  void itoa(int num, char *s) {
7     if (num < 0) { // Se o valor for negativo.
8        *s++ = '-';
9        n = -n;
10    }
11    itoa_aux( num , s );
12   *s = 0; // Terminar a String
13 )
```