Funções Recursivas

Funções recursivas

O exemplo mais comum:

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

Versão sem recursão:

```
int fat(int n){
  int i = 0; int f = 1;
  for(i=1; i <=n; i++) f *= i;
  return f;
}
```

Outro exemplo

Série de Fibonacci (0,1,1,2,3,5,8,13,...) :

```
int fib(int n){
  if(n <= 1) return m;
  return fib(n-1)*fib(n-2);
}
```

Versão sem recursão:

```
int fibb(int n){
  int f1 = 0, f2 = 1, f3, i;
  for(i=1; i <= n; i++) {
    f3 =  f2 + f1;
    f1 = f2; f2 = f3;
  }
  return f1;
}
```

Mais um exemplo

Torres de Hanoi

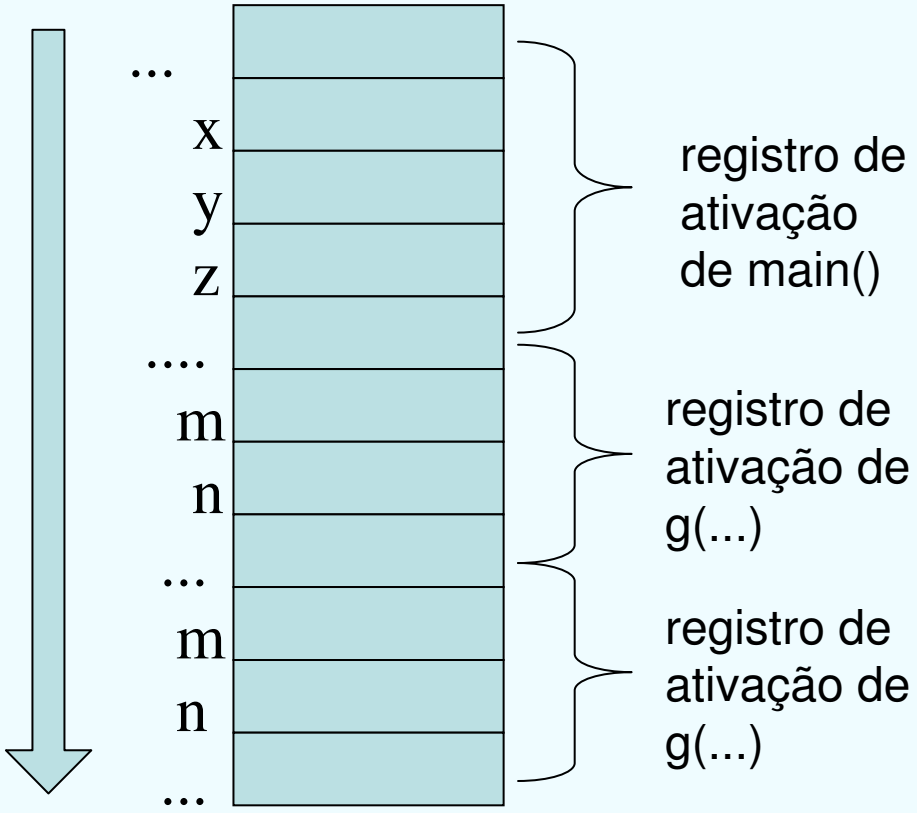
```
void moveTorre(int n, char a, char b, char c){
  if(n > 0) {
    moveTorre(n-1,a,c,b);
    printf("mover de %c para %c\n",a,b);
    moveTorre(n-1,c,b,a);
  }
}
```

Eliminação da recursão

- Dada uma função recursiva, sempre é possível escrever uma função equivalente, sem recursão.
- Para se eliminar a recursão, na maioria das vezes é necessário recorrer ao uso de uma pilha.
- Caso simples: a função faz uma única chamada recursiva durante sua execução (ex. fatorial).

Um exemplo

```
#include <stdio.h>
void g(...){
  char m,n;
  ...
  g(...);
  ...
  g(...);
  ...
}
int main(){
  char x,y,z;
  ...
  g();
  ...
}
```



Cada ativação de g() usa o seu próprio registro de ativação que contém suas variáveis locais e parâmetros.

Eliminação da recursão
Idéia Geral (usando uma pilha)

- Cada chamada recursiva é substituida por comandos para
  - empilhar o valor das variáveis locais e parâmetros
  - alterar os valores das variáveis locais e parâmetros
  - voltar ao início da função
- O encerramento de uma chamada (retorno) é substituído por comandos para
  - desempilhar os valores dos parâmetros e variáveis locais.
  - continuar a execução a partir do ponto onde seria o retorno da chamada recursiva correspondente (esse 'ponto' pode ser representado por um valor adicional na pilha).

Eliminação da recursão
Idéia Geral (usando uma pilha)

- Na prática, ao se usar uma linguagem de programação de alto nível, ‘voltar ao início da função’ e ‘continuar a execução’ a partir determinado ponto nem sempre são óbvios.
- Em geral é necessário recorrer a combinações de comandos ‘while’ e ‘switch’.

Um exemplo (1)

```
typedef enum { iniciar, imprimir, encerrar } operacao;

void moveTorre2(int n, char a, char b, char c){
  empilha(encerrar,-1,'?','?','?');
  operacao op = iniciar;
  do{
    switch(op){
      case iniciar:  if(n > 0) {
                       empilha(iniciar, n-1,c,b,a);
                       empilha(imprimir,n-1,a,b,c);
                       empilha(iniciar, n-1,a,c,b);
                     }
                     break;
      case imprimir: printf("mover de %c para %c\n",a,b);
                     break;
    }
    desempilha(&op,&n,&a,&b,&c);
  }while(op != encerrar);
}
```

Um exemplo (2)

- **Pilha em alocação sequencial (vetor)**

```
#define PMAX 100
#define EMPTY -1

int pilha[PMAX];
int t;

void erro(char* s){
  printf("erro: %s\n",s);
  exit(-1);
}

void iniciaPilha(){
  t = EMPTY;
}
```

Um exemplo (3)

```
void empilha(operacao op, int n, char a, char b, char c){
  if(t+5 >= PMAX) erro("Pilha cheia!");
  pilha[++t] = (int) op;
  pilha[++t] = n;
  pilha[++t] = (int)a;
  pilha[++t] = (int)b;
  pilha[++t] = (int)c;
}

void desempilha(operacao* op, int* n, char* a, char* b, char* c){
  if(t-5 < EMPTY) erro("Pilha vazia!");
  *c  = (char)pilha[t--];
  *b  = (char)pilha[t--];
  *a  = (char)pilha[t--];
  *n  = pilha[t--];
  *op = (operacao)pilha[t--];
}
```

Um exemplo (4)

- **Implementação de pilha usando alocação ligada**

```
typedef struct config* pconfig;
typedef struct config {
                  operacao opp;
                  int nn;
                  char aa,bb,cc;
                  pconfig prox;
                } config;

pconfig topo;

void iniciaPilha(){
  topo = NULL;
}
```

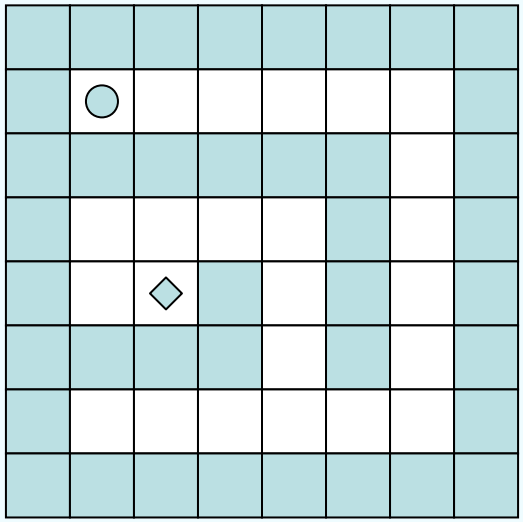
Um exemplo (5)

```
void empilha(operacao op, int n, char a, char b, char c){
  pconfig p = (pconfig)malloc(sizeof(config));
  if(p == NULL) erro("Falha ao alocar memória");
  p->opp = op;     p->nn = n;
  p->aa = a;       p->bb = b;
  p->cc = c;
  p->prox = topo;  topo = p;
}

void desempilha(operacao* op, int* n, char* a, char* b, char* c){
  pconfig p = topo;
  if(p == NULL) erro("Pilha vazia");
  *op = p->opp;  *n  = p->nn;
  *a  = p->aa;   *b  = p->bb;
  *c  = p->cc;
  topo = p->prox;
  free(p);
}
```

Recursão e retrocesso

- Várias alternativas devem ser ‘tentadas’
- Se uma alternativa não é bem sucedida, voltar ao ‘estado anterior’ e tentar a próxima alternativa.
- Exemplo: Procurar um caminho num labirinto

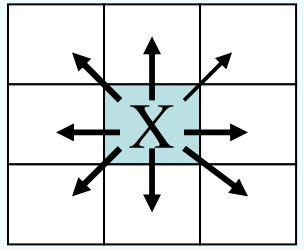


Recursão e retrocesso

- Idéia geral:

```
void tenta(alternativa){
  if(<alternativa viável> {
     <marca alternativa selecionada>
     if(<atingiu objetivo>) <encontrou solução>;
     else <para todas alternativas alt atuais>
          tenta(alt);
     <‘desmarca’ alternativa selecionada>
  }
}
```

Um exemplo (labirinto)




```
/* matriz que define o 'labirinto'*/
int m[8][8];

void tenta(int i, int j){
  int dx[8] = {-1,-1,-1,0,0,1,1,1};
  int dy[8] = {-1,0,1,-1,1,-1,0,1};
  int k;
  if(m[i][j] == ' '){        /* alternativa viável          */
    m[i][j] = 'x';           /* marcar o caminho            */
    if((i==DI)&&(j==DJ))     /* atingiu o objetivo          */
        imprime();
    else {                   /* tentar todas alternativas */
      for(k=0; k < 8; k++) tenta(i+dx[k],j+dy[k]);
    }
    m[i][j] = ' ';           /* desfazer a marca            */
  }
}
```

Um exemplo (labirinto)

```
/* inicia a matriz m */
void inicia(){
  int i,j;
  for(i=0; i < 8; i++)
    for(j=0; j < 8; j++) m[i][j] = ' ';
  for(i=0; i < 8; i++){  /* 'bordas' do labirinto */
    m[i][0] = '#';
    m[i][7] = '#';
    m[7][i] = '#';
    m[0][i] = '#';
  }
  for(i=0; i < 6; i++){ /* 'paredes' internas */
    m[2][i] = '#';
    m[4][i+2] = '#';
  }
  m[5][2] = '#';
  m[5][4] = '#';
}
```