Recursividade

Aula 17

Recursão

- É um conceito geral (independe da computação)
  - Letras: GNU, Allegro (Allegro Low-Level Game Routines)
  - Matemática
  - Artes: música, pintura
- **É a definição da solução de um problema em função de uma instância menor dele mesmo!**

Exemplo de definição recursiva

- Fatorial de um número:
  - 5! = 5*4*3*2*1
  - 4! = 4*3*2*1
  - 5! = 5*4!

- Podemos dizer que fatorial(n) = n*fatorial(n-1)!

Forma Recursiva

- A recursão precisa de três elementos base:
  - Caso base ou condição de parada
  - Processamento a cada etapa
  - Chamada recursiva

Exemplo

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

```
int main(){
    int n;
    scanf("%d", &n);
    printf("%d\n", fatorial(n));
    return 0;
}
```

- Condição de Parada
- Processamento
- Chamada recursiva

Recursividade x Iteratividade

- Muitas vezes, a forma recursiva é mais natural para abordamos um problema
  - Série de Fibonacci

$$f(n) = \begin{cases} n = 0 & 0 \\ n = 1 & 1 \\ n > 1 & f(n-1) + f(n-2) \end{cases}$$

Fibonacci Iterativo

```
#include <stdio.h>
int main( void )
{
   int i, j, n, fib;
   scanf( "%i", &n );
   fib = 0;
   j = 1;
   for(i=1; i<n; i++)
   {
       j = fib + j;
       fib = j - fib;
   }
   printf(“%d\n”, fib);
   return 0;
}
```

Fibonacci Recursivo

```
int fib( int x )
{
   if( x <= 1 )
       return x;
   else
       return fib(x-1) + fib(x-2);
}

int main(){
      int n;
      scanf(“%d”, &n);

      printf(“%d\n”, fib(n));
      return 0;
}
```

Explorando as possibilidades

- A recursão pode ser usada como procedimento

- Normalmente usamos recursão quando queremos explorar todas as possibilidades de um problema
  - Backtracking

Processamento

- O processamento de informações pode ocorrer:
  - antes da chamada recursiva (pré-processamento)
  - após a chamada recursiva (pós-processamento)

- Uma recursão que faz todo o processamento antes da chamada recursiva é uma recursão em cauda (similar a um loop)

Exemplo - Conversão de Base

- Faça um programa que converta um número da base decimal para a base binária
  - Ex: 15 = 1111

Solução Recursiva

- Para n = 10
  - 10 / 2 = 5 (Resto 0)
  - 5 / 2 = 2 (Resto 1)
  - 2 /2 = 1 (Resto 0)
  - 1 / 2 = 0 (Resto 1)

- O valor binário é a ordem invertida dos restos: 1010

Solução Recursiva

- A divisão é um pré-processamento (você divide e então faz a chamada)

- A impressão é um pós-processamento (você só imprime o número após a chamada da função retornar)

Solução Recursiva

```
#include <stdio.h>

void binario(int n){
     if(n > 0){
          binario(n/2);
          printf("%d", n%2);
     }
}
```

```
int main(){
     int n;

     scanf("%d", &n)
     binario(n);
     printf("\n");

     return 0;
}
```

Recursão Mútua ou Indireta

- Uma recursão pode ser montada usando mais de uma função

- Nesse caso, a recursão se dá quando um ciclo de chamadas é feito
  - Ex: função1 → função2 → função 1

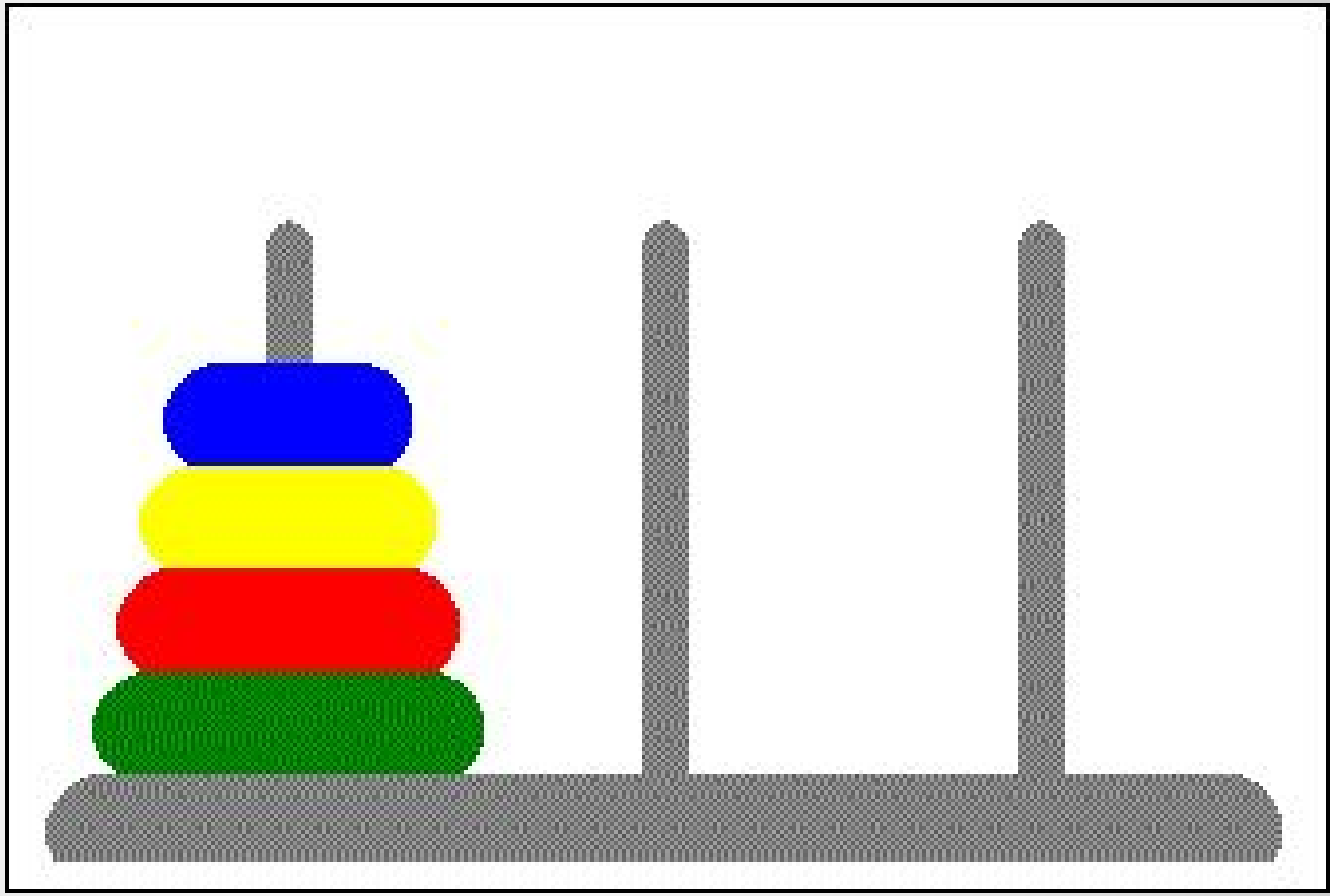
Exemplo

```
int ehPar(int n){
    if(n == 0)
        return 1;
    else
        return ehImpar(n-1);
}
```

```
int ehImpar(int n){
    if(n == 0)
        return 0;
    else
        return ehPar(n-1);
}
```

Dúvidas?

Torres de Hanói



Torres de Hanói

- Mover os discos de uma torre para a outra
- Um disco maior não pode ser colocado sobre um disco menor

Solução Recursiva

```
int main(){
     int n;

     printf("Digite o numero de discos a serem movidos: ");
     scanf("%d", &n);

     //Considerando os 3 pinos respectivamente A, B e C
     resolve_hanoi(n, 'A', 'B', 'C');

     return 0;
}
```

Solução Recursiva

```
#include <stdio.h>

void resolve_hanoi(int n, char origem, char inter, char
destino){
    if(n>0){
        resolve_hanoi(n-1, origem, destino, inter);
        printf("Mova %d de %c para %c\n", n, origem, destino);
        resolve_hanoi(n-1, inter, origem, destino);
    }
}
```

Problema 01

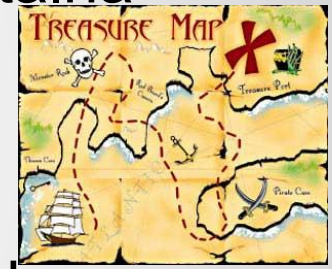
- Zoro é um grande espadachim, porém ele tem um péssimo senso de direção! Isso faz com que ele constantemente se perca dos outros e acabe indo parar no lugar errado dos duelos.

Mapa do (T?)Zoro

- Preocupados que isso possa afetar a sua equipe, o bando do Chapéu de Palha resolveu pedir que os alunos de ITP fizessem um programa que ajudasse o Zoro a achar os locais corretos das batalhas!

Zoro Total?

- Seu programa lerá quatro valores inteiros: a posição (x,y) inicial de Zoro e a posição (x,y) onde a batalha acontece.



- Você deverá traçar um mapa para que o zoro chegue da posição inicial até a posição final, imprimindo cada passo que Zoro deve dar (senão, ele irá se perder)!

Exemplo de Entrada e Saída

- Entrada

  0 0 5 0

Saída

Passa por (0,0)
Passa por (1,0)
Passa por (2,0)
Passa por (3,0)
Passa por (4,0)
Chega em (5,0)

Proposta de Solução

```
void caminho(int origx, int origy, int* destx, int* desty){
     if(origx < 0 || origy < 0)
          return;
     else if(origx != *destx){
          if(origx > *destx){
               printf("Passa por (%d,%d)\n", origx, origy);
               caminho(--origx, origy, destx, desty);
          }
          else{
               printf("Passa por (%d,%d)\n", origx, origy);
               caminho(++origx, origy, destx, desty);
          }
     }
```

Proposta de Solução

```
else if(origy != *desty){
     if(origy > *desty){
          printf("Passa por (%d,%d)\n", origx, origy);
          caminho(origx, --origy, destx, desty);
     }
     else{
          printf("Passa por (%d,%d)\n", origx, origy);
          caminho(origx, ++origy, destx, desty);
     }
}
else{
     printf("Chega em (%d,%d)\n", origx, origy);
}}
```

Proposta de Solução

```
int main(){
     int iniciox, inicioy;
     int finalx, finaly;

     scanf("%d %d %d %d", &iniciox, &inicioy, &finalx, &finaly);
     caminho(iniciox, inicioy, &finalx, &finaly);

     return 0;
}
```

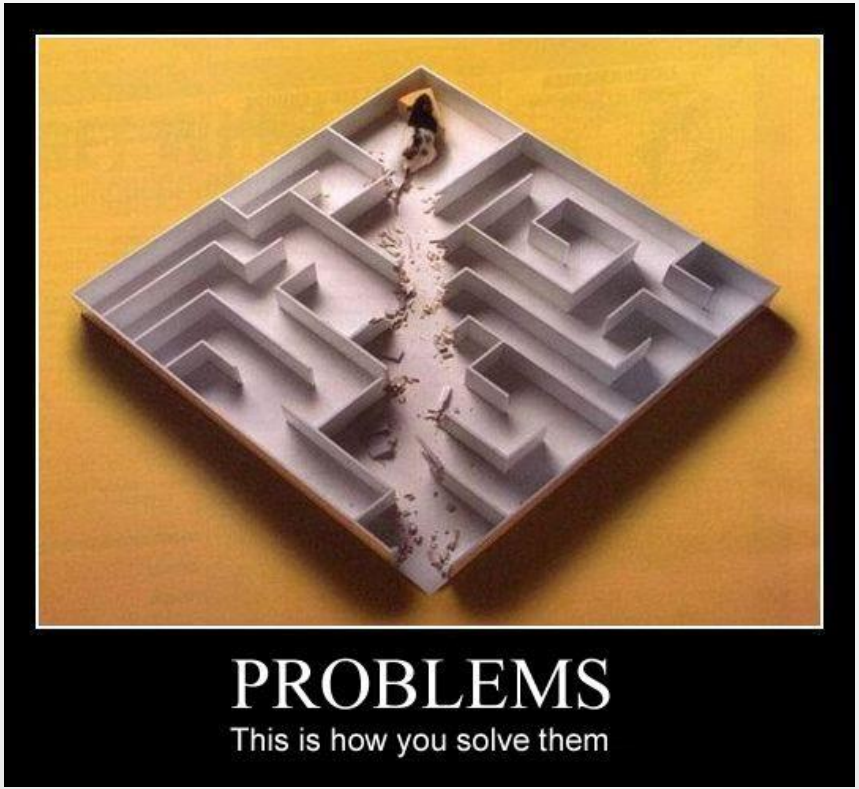
Problema 02

- Dado um labirinto, descubra se é possível sair da posição inicial e chegar em uma determinada posição final.
- Seu programa lerá um mapa NxN (N <= 40). Cada posição do mapa será representado por um valor inteiro, 0 sendo uma casa livre para andar e 1 sendo uma barreira
- Verificar se a partir de uma posição inicial (x,y) é possível chegar em uma posição final (x', y').

Maze Runner

- Seu programa lerá da entrada:
  - A dimensão do mapa N
  - NxN entradas, representando cada posição do labirinto
  - A posição inicial
  - A posição final desejada
- Seu programa deve imprimir “Freedom” se conseguir alcançar a posição, e “Trapped” caso contrário.

Maze Runner



Solução

- Vamos usar uma técnica mais avançada de programação para resolver esse problema: uma função que explora todas as possibilidades (backtracking), mas se lembra do caminho que fez até o ponto em que está.

- Dessa forma iremos evitar que fiquemos presos em um ciclo dentro do labirinto!

Solução - Função Principal

```
#include <stdio.h>
#include <string.h>

int main(){
    int lab[41][41];
    int percurso[41][41];
    int n, ok, i, j;
    int origx, origy, destx, desty;
    memset(lab, 0, sizeof(lab));
    memset(percurso, 0, sizeof(percurso));
```

Solução - Função Principal

```
scanf("%d", &n);

 for(i=0; i<n; i++){
     for(j=0; j<n; j++){
         scanf("%d", &lab[i][j]);
     }
 }

 scanf("%d %d", &origx, &origy);
 scanf("%d %d", &destx, &desty);
```

Solução - Função Principal

```
   ok = percorreLab(origx, origy, destx, desty, n-1, lab,
percurso);

    if(ok)
        printf("Freedom\n");
    else
        printf("Trapped\n");

    return 0;
}
```

Função de Movimento - Caso Base

```
int percorreLab(int ox, int oy, int dx, int dy, int tam, int
mat[41][41], int path[41][41]){
    int ok;

    if(ox == dx && oy == dy)
        return 1;
```

Testa o movimento para baixo

```
   else{
        if(ox < tam && mat[ox+1][oy] == 0 && path[ox+1][oy] ==
0){
            path[ox+1][oy] = 1;
            ok = percorreLab(ox+1, oy, dx, dy, tam, mat, path);
            if(ok) return 1;
            path[ox+1][oy] = 0;
        }
```

Testa o movimento para direita

```
if(oy<tam && mat[ox][oy+1] == 0 && path[ox][oy+1] == 0){
     path[ox][oy+1] = 1;
     ok = percorreLab(ox, oy+1, dx, dy, tam, mat, path);
     if(ok) return 1;
     path[ox][oy+1] = 0;
 }
```

Testa o movimento para esquerda

```
if(oy > 0 && mat[ox][oy-1] == 0 && path[ox][oy-1] == 0){
     path[ox][oy-1] = 1;
     ok = percorreLab(ox, oy-1, dx, dy, tam, mat, path);
     if(ok) return 1;
     path[ox][oy-1] = 0;
 }
```

Testa o movimento para cima

```
        if(ox > 0 && mat[ox-1][oy] == 0 && path[ox-1][oy] == 0){
             path[ox-1][oy] = 1;
             ok = percorreLab(ox-1, oy, dx, dy, tam, mat, path);
             if(ok) return 1;
             path[ox-1][oy] = 0;
         }

         return 0;
    }
}
```