Laboratório 12

Variáveis globais e estáticas;
Argumentos para a main();
Operações com bits.

Oficina de Programação em C (ICP037)

Prof. Ronald Souza
IC/UFRJ

Objetivo

Praticar os conceitos vistos na Aula 12.

Atividade 1: Copie o código abaixo (Sequência de Fibonacci - **continua na próxima página!**).

a) Inclua um inteiro como **parâmetro de entrada da main(),** correspondente ao tamanho da sequência a ser calculada. **Defenda** o tamanho para N <= 50.
b) Imprima na main() o total de cálculos de cada método (salvo em suas respectivas variáveis estáticas!)
c) Há diferença significativa de tempo entre os dois métodos? Por que?

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 50

long long valores[MAX] = { 0 }; //Global

long long fib(int n) {
    static int calculos = 0;
    calculos++;
    return (n == 1 || n == 0) ? n : fib(n - 1) + fib(n - 2);
}
long long smartFib(int n) {
    static int calculos = 0;
    if (n == 1 || n == 0)
       return n;

    if (!valores[n]) {
       valores[n] = smartFib(n - 1) + smartFib(n - 2);
       calculos++;
    }
    return valores[n];
}
```

```
int main() {
    int n, f1, f2;
    double tempoGasto;
    clock_t inicio;
    //RECEBER n pela main!!
    inicio = clock(); //Inicia contagem de tempo
    f1 = fib(n);
    tempoGasto = ((double)(clock() - inicio)) / CLOCKS_PER_SEC;
    printf("Fib(%d) = %d (calculado em %.2fs)\n", n, f1, tempoGasto);

    //Alocação dinâmica de um vetor de inteiros de 64 bits:
    //long long* valores = (long long*)calloc(MAX, sizeof(long long));
    inicio = clock();
    f2 = smartFib(n);
    tempoGasto = ((double)(clock() - inicio)) / CLOCKS_PER_SEC;
    printf("smartFib(%d) = %d(calculado em %.2fs)\n", n, f2, tempoGasto);
    return 0;
}
```

Atividade 2: Escreva uma função em C que receba um inteiro como entrada e retorne 1 se ele for par e 0 se ele for ímpar, **sem usar os operadores de divisão e resto.**

Atividade 3: Copie o código abaixo e o execute. Note que ele já possui as versões bitwise dos operadores relacionais “>=” (maior que ou igual a) e “<” (menor que).
→ Com base nos exemplos, **implemente a versão “bitwise” dos demais operadores relacionais**, isto é, ”<=” (menor ou igual), “>” (maior que) e “==” (igual).

```
#include <stdio.h>
#define TAM 3
int main() {
      int vet1[TAM] = { 1,10,100 };
      int vet2[TAM] = { 2,10,1 };
      int gate = 0;     //Nossa "válvula" - TODOS os bits em 0 ou em 1 SEMPRE.
      int s = 0;        //somatório (acumulador).
      for (int i = 0; i < TAM; i++) {
            //O código abaixo equivale a "if (vet1[i] >= vet2[i]) s += vet1[i]":
            gate = ~((vet1[i] - vet2[i]) >> 31);
            s += gate & vet1[i];
      }
      printf("\nMAIOR OU IGUAL\nExpectativa: %d\nRealidade = %d\n", 110, s);

      s = 0; //Zeramos o somatório
      for (int i = 0; i < TAM; i++) {
            //O código abaixo equivale a "if (vet1[i] < vet2[i]) s += vet1[i]":
            gate = (vet1[i] - vet2[i]) >> 31;
            s += gate & vet1[i];
      }
      printf("\nMENOR\nExpectativa: %d\nRealidade = %d\n", 1, s);
      //IMPLEMENTAR OS DEMAIS CASOS!
      return 0;
}
```