O Maravilhoso Visualg

Mário Leite

...

O processo de criação de um programa passa, necessariamente, pela análise do problema a ser resolvido e em seguida a sua solução deve ser criada com uma sistemática que possa resolver o problema. Essa sistemática é baseada numa lógica que deve implementar alguma solução. Mas, embora a linguagem de programação seja o processo final para automatizar a solução do problema, o programa tem que ser elaborado antes, de modo que a solução seja a melhor possível. E é importante frisar que *codificar* (criar o código-fonte na linguagem de programação) é apenas uma questão de aplicar corretamente a sintaxe da linguagem; mas, isto não resolve o problema, pois o fundamental é achar a solução mais eficiente e mais eficaz, através da lógica de programação, que pode ser definida como uma "*técnica de desenvolver sequências de ações para atingir um objetivo bem determinado*". As sequências são adaptadas para uma linguagem de computador a fim de produzir o programa, onde o usuário apresenta o problema ao programador, que o refina e traduz para que o computador possa executar as ordens embutidas nesses refinamentos. E para que essas ordens sejam executadas corretamente elas devem ser escritas como texto em uma sequência lógica, tal que a solução apresentada pelo computador seja correta e satisfaça o usuário. As ordens são traduzidas num texto chamado "*código-fonte*", criadas pelo programador que as escreve e as insere no computador para que apresente uma solução para o problema, a partir originalmente, seguindo uma escrita informal denominada "algoritmo". A **figura 1** mostra um esquema de como fica a interação: *usuário-programador-computador*, na sequência real do cenário.

Então, o PRINCIPAL no desenvolvimento de qualquer sistema computacional passa, primeiramente, pela etapa de "criação do algoritmo" para, só DEPOIS, criar o código em uma linguagem real de programação. Entretanto, esse algoritmo pode ser automatizado de maneira a ser testado durante a sua criação; é aí que entra o **Visualg**: uma ferramenta de auxílio no aprendizado de programação, genuinamente brasileira, e adotada em muitas instituições de ensino. Esta ferramenta foi criada, pelo professor Cláudio Morgado de Souza; depois o professor Antonio Carlos Nicolodi assumiu o seu desenvolvimento, com a versão 3.0. A **figura 2** mostra o IDE dessa ferramenta, na sua versão 3.0. A **figura 3** mostra a saída do programa codificado em C++ e a **figura 4** a saída do mesmo programa codificado em Visualg. Observe que o tempo gasto com a linguagem C++ para mostrar os quatro primeiros números perfeitos foi infinitamente menor (não chegou nem a **1** segundo), contra os quase **14** minutos com o Visualg. Então, a dedução lógica seria a seguine: "O C++ É MELHOR QUE O VISUAL!"; não necessariamente, pois quando se trata de APRENDER programação, como foi dito acima, a solução do problema passa, necessariamente, pela criação do algoritmo. E isto pode ser feito com uma ferramenta computacional; e é nessa etapa que se cria a solução do problema, que não tem nada a ver com sua codificação. Criar a solução de um problema diretamente em uma linguagem tão complicada e difícil de aprender como o C++ assusta qualquer pretendente a programador; por isto o **Visualg** é uma maravilha de ferramenta para o aprendizado...

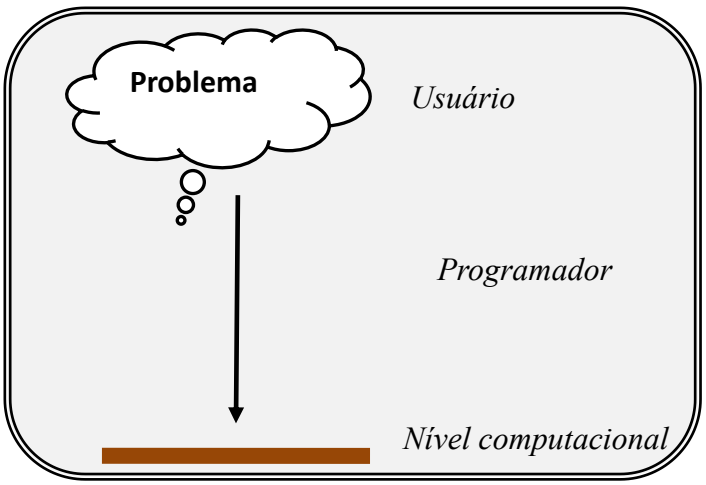


Figura 1 - Trazendo o problema para o nível computacional

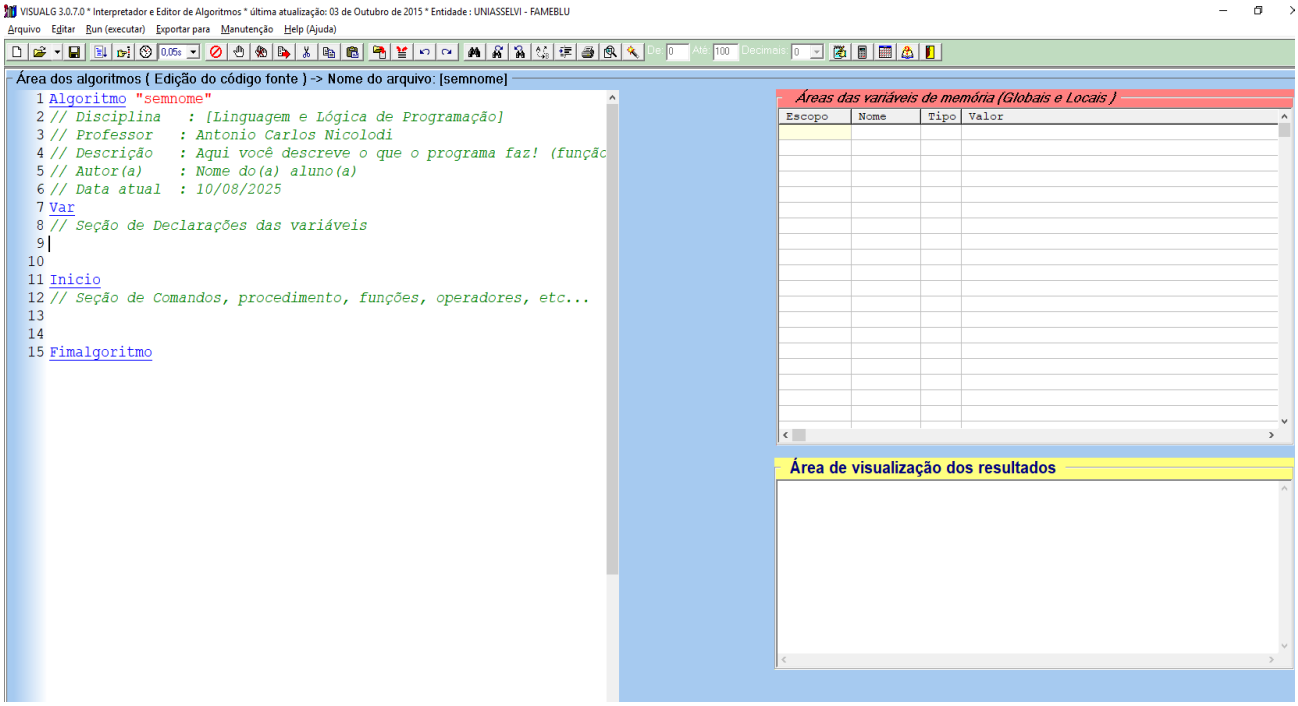
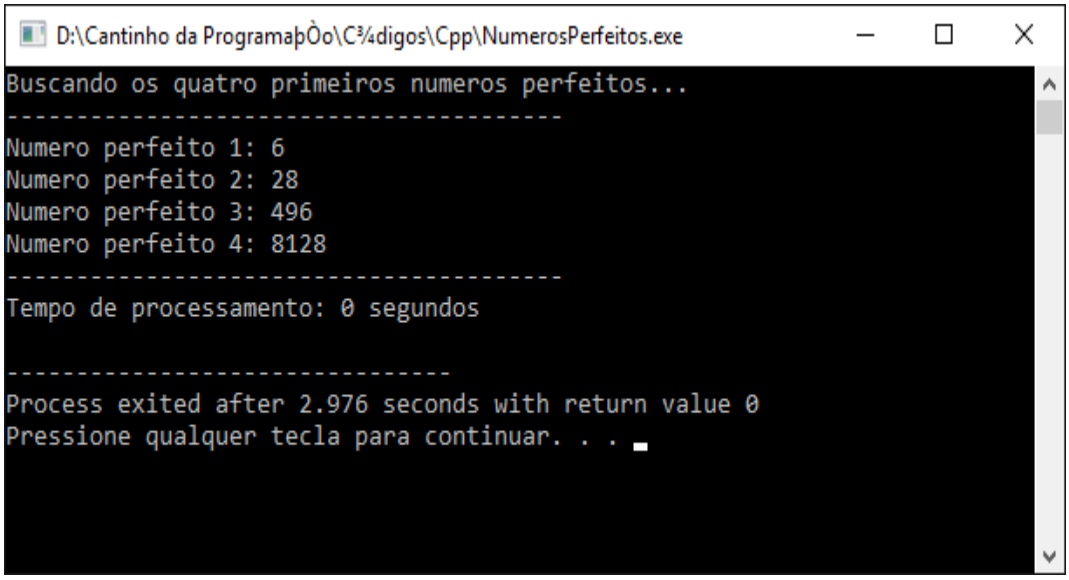
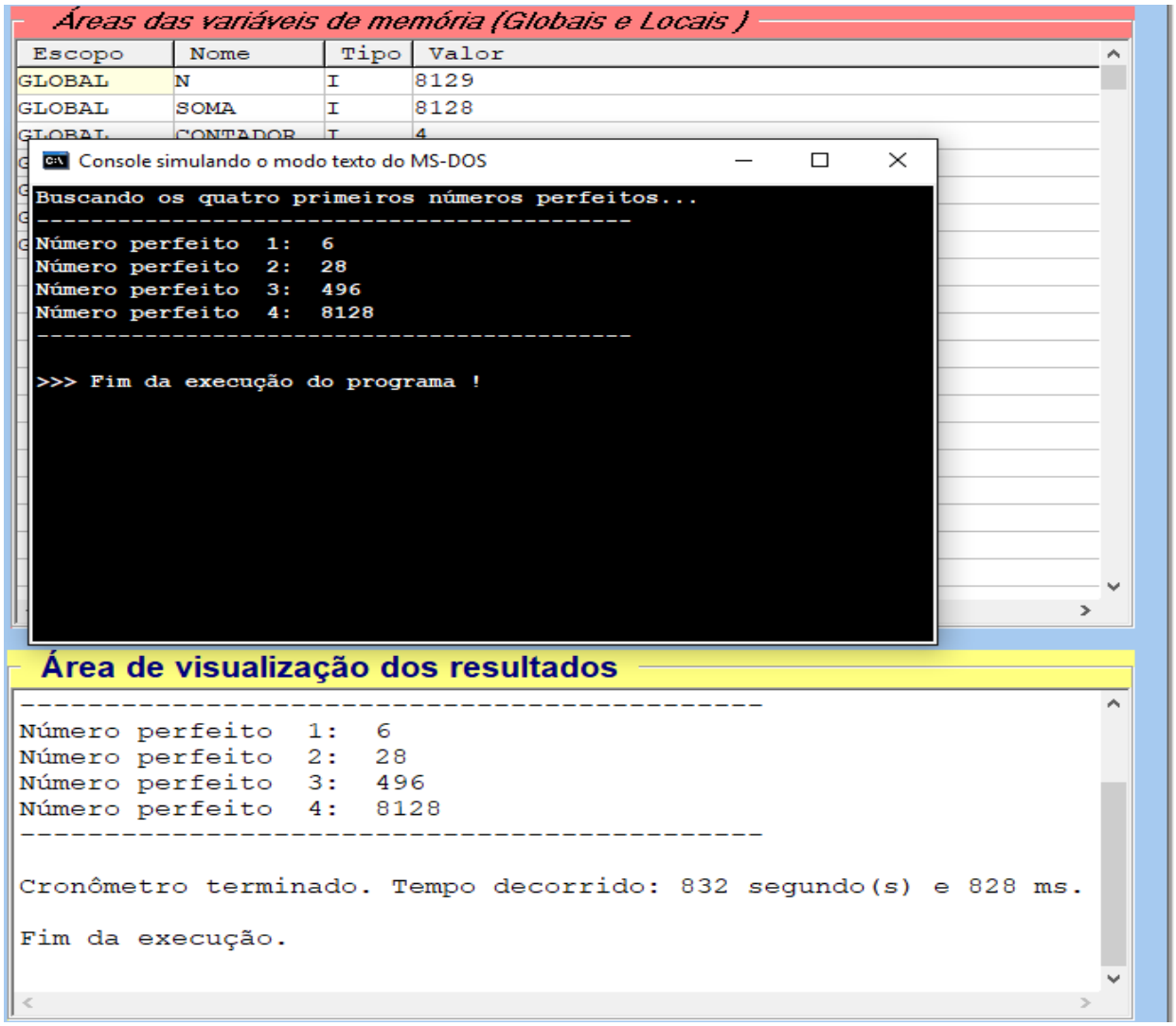


Figura 2 - O ambiente de desenvolvimento do Visualg



‘ **Figura 3 - Saída do programa codificado em C++**



‘ **Figura 4 - Saída do programa codificado em Visualg**

```cpp
//NumrosPerfeitos.cpp

#include <iostream>
#include <cmath>
#include <ctime>

using namespace std;

bool isPrime(long long num) {
    if (num <= 1) return false;
    if (num == 2) return true;
    if (num % 2 == 0) return false;

    for (long long i = 3; i * i <= num; i += 2) {
        if (num % i == 0) return false;
    }
    return true;
}

long long powerOfTwo(int exp) {
    return (long long)1 << exp; // Calcula 2^exp usando shift bitwise
}

long long CalcularNumeroPerfeito(int p) {
    long long mersenne = powerOfTwo(p) - 1;
    return powerOfTwo(p - 1) * mersenne;
}

int main() {
    cout << "Buscando os quatro primeiros numeros perfeitos..." << endl;
    cout << "----------------------------------------" << endl;

    clock_t inicio = clock();

    int encontrados = 0;
    int p = 2;

    while (encontrados < 4) {
        if (isPrime(powerOfTwo(p) - 1)) {
            long long perfeito = CalcularNumeroPerfeito(p);
            encontrados++;
            cout << "Numero perfeito " << encontrados << ": " << perfeito << endl;
        }
        p++;
    }

    clock_t fim = clock();
    double tempo = (double)(fim - inicio) / CLOCKS_PER_SEC;

    cout << "----------------------------------------" << endl;
    cout << "Tempo de processamento: " << tempo << " segundos" << endl;

    return 0;
}
```

```
Algoritmo "NumerosPerfeitos"
var
   n, soma, contador, i: inteiro
   inicioTempo, fimTempo: real
   tempoDecorrido: real

Inicio
   //Inicia a contagem do tempo de processamento
   Cronometro on
   contador <- 0
   n <- 2  // Começamos a verificar a partir do número 2

   Escreval("Buscando os quatro primeiros números perfeitos...")
   Escreval("-------------------------------------------")

   Enquanto (contador < 4 Faca)
      soma <- 1  // Todo número é divisível por 1

      // Verificamos os divisores próprios de n
      Para i De 2 ate n div 2 Faca
         Se (n mod i = 0) Entao
            soma <- soma + i
         Fimse
      FimPara

      //Se a soma dos divisores for igual ao número, é perfeito
      Se (soma = n) Entao
         contador <- contador + 1
         Escreval("Número perfeito ", contador, ": ", n)
      Fimse

      n <- n + 1
   FimEnquanto
   Escreval("-------------------------------------------")
   //IniFinalizacia a contagem do tempo de processamento
   Cronometro off
FimAlgoritmo
```