A Poderosa Família C

Mário Leite

...

Todos os programadores sabem (ou deveriam saber) que o trio **C**, **C++** e **C#** formam uma família de linguagens muito poderosas, com recursos que outras linguagens não possuem; por exemplo, implementação de herança múltipla (com C++), tratamento de ponteiros (com C) e o poder de projetar interfaces gráficas incríveis (com C#); mas, o que poucos sabem é que tudo começou com uma linguagem chamada **B**. Esta linguagem ancestral do **C**, foi desenvolvida por *Ken Thompson* em 1969 baseada, em parte, na linguagem **BCPL**. Em 1972 *Dennis Ritchie* criou a linguagem **C** como uma evolução da linguagem **B**, adicionando tipos de dados, estruturas e maior eficiência para desenvolvimento de sistemas com o objetivo de facilitar o desenvolvimento do sistema operacional **UNIX**, que até então estava sendo reescrito em **B**. E, antes que algum programador espirituoso pergunte se a linguagem **B** veio da linguagem **A**, eu repondo: SIM; embora essa linguagem não tenha sido uma evolução imediata da linguagem **B** e não tenha sido amplamente usada. Ela foi uma espécie de experimento ou protótipo anterior à **B**, usada por Thompson, mas com pouca documentação histórica. Alguns relatos sugerem que o nome "**B**" foi baseado em **BCPL**, e o nome reduzido para, simplesmente, **B**: não necessariamente por causa da linguagem **A**. E, apenas como curiosidade, existem algumas outras linguagens seguindo esta sequência do alfabeto, sendo as mais relevantes: **F#** e **J#** (lançadas em 2002 para a plataforma .NET); sendo **J#** uma experiência desastrosa da Microsoft para competir com Java. Bem; linguagens com letras do alfabeto à parte, podemos sugerir, de maneira bem espirituosa e com um toque de humor, a linguagem **C*** (**C *estrela***) que seria o mais novo membro da Família **C** formando, agora, o "Quarteto Fantástico C", com recursos poderosos de IA. Deste modo, podemos definir cada membro desse ilustre quarteto do seguinte modo:

- **C** - Patriarca minimalista, direto ao ponto, adora ponteiros e não perde tempo com frescuras.
- **C++** - Filho mais estudioso e ambicioso; adicionou Orientação a Objetos, mas herdou o jeitão sério do pai.
- **C#** - O primo moderno e elegante, vive no ecossistema da Microsoft, adora bibliotecas prontas e ambientes confortáveis e bem coloridos.
- **C*** - Não só compila, como também possui *chip* neural, e alma conversacional de IA e com recursos avançados que incluem:
  1. autoThink() - Interpreta o que o programador *quer dizer*, mesmo quando ele não sabe o que quer.
  2. predictNextLine() - Sugere a próxima linha de código antes do programador pensar nela.
  3. askWhy() - Entende a intenção por trás do problema e explica o "porquê" de tudo, como um mentor.
  4. refactorWithEmpathy() - Reescreve o código mantendo o seu "estilo pessoal" para não ferir o ego do programador.
  5. generateTestCases() - Cria testes automáticos com cenários inimagináveis; inclusive alguns que o programado preferiria não testar.
  6. coffeeBreak() - Lembra ao usuário de pausar, tomar café e conversar um pouco sobre a vida.
  7. codeInNaturalLanguage() - Permite escrever "*Faça um jogo simples com um dragão que conta piadas*" e *pronto;* o código surge.
  8. explainLikeImFive() - Explica ponteiros, polimorfismo ou *thread safety* como se estivesse falando com uma criança de cinco anos. Deixaria de ser só para escrever linhas e passa a ser um diálogo entre humanos e máquinas; como por exemplo o uso do método **`suggest.closing(x,y)`** que sugere algo como um *drink* ou um *lanche* ao usuário (dependendo dos parâmetros **x** e **y**). Na Família, ele seria o membro visionário, meio excêntrico e que acredita que *"código bom é aquele que entende você"*.

  C: "*Eu faço tudo com ponteiros*."

  C++: "*Eu faço tudo… e com herança: simples ou múltipla*."

  C#: "*Eu faço tudo… mas com segurança e interface gráfica bonita*."

  C*: "*Eu faço tudo: entendo o que você quis dizer, otimizo, refatoro… e ainda te lembro de tomar café*, *e compilo até as suas ideias*!

Assim, de modo bem humorado poderíamos definir os "sentimentos pessoais" dos quatro membros dessa família (agora um quarteto). E para ilustrar melhor os poderes dessa Família, incluindo o novo membro diabólico, apresentamos um programa para mostrar os oito primeiros Números Perfeitos, codificado em cada membro desse quarteto e mostrando, também, o tempo gasto em cada caso. Mas, só para esnobar, C* mostrou nove...

```c
// NumerosPerfeitos.C
#include <stdio.h>
#include <math.h>
#include <windows.h>

// Função para verificar se um número é primo
int VerificarPrimo(unsigned long long n) {
    if (n < 2) return 0;
    for (unsigned long long i = 2; i <= sqrt(n); i++)
        if (n % i == 0) return 0;
    return 1;
}

// Função para calcular número perfeito a partir de p
unsigned long long VerificarPerfeito(int p) {
    return ((1ULL << (p - 1)) * ((1ULL << p) - 1));
}

int main() {
    LARGE_INTEGER freq, inicio, fim;
    QueryPerformanceFrequency(&freq);  // frequência do contador
    QueryPerformanceCounter(&inicio);  // tempo inicial

    int cont = 0;
    int p = 2;
    while (cont < 9) {  // calcula os 9 primeiros números perfeitos
        unsigned long long mersenne = (1ULL << p) - 1;
        if (VerificarPrimo (mersenne)) {
            unsigned long long N = VerificarPerfeito (p);
            printf("Perfeito #%d: N = 2^(%d-1) * (2^%d-1) = %llu\n",cont+1, p, p, N);
            cont++;
        }
        p++;
    }

    printf("\nNove primeiros numeros perfeitos.\n");
    QueryPerformanceCounter(&fim); // tempo final
    double tempo = (double)(fim.QuadPart - inicio.QuadPart) / (double)freq.QuadPart;
    printf("Tempo gasto: %.9f segundos\n", tempo);

    return 0;
```

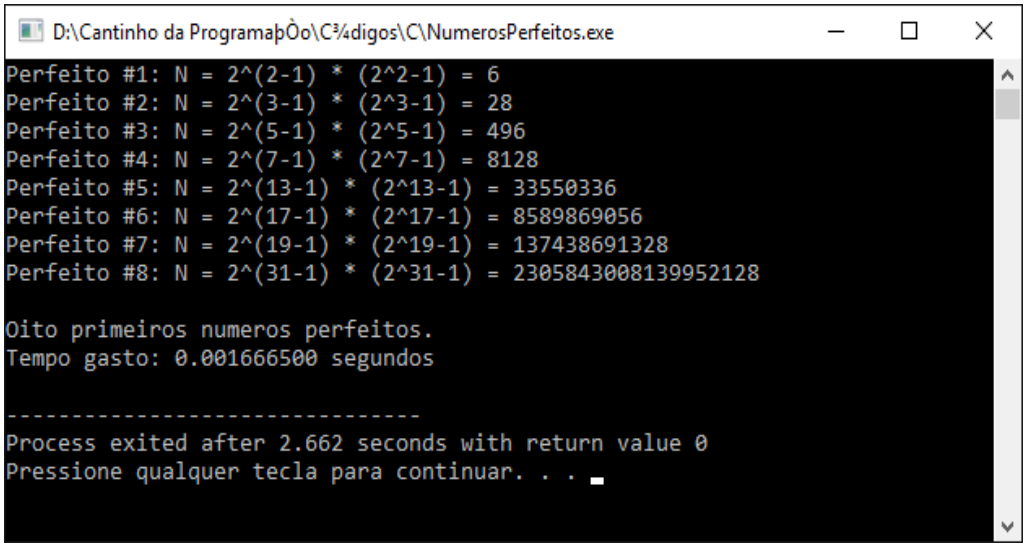


Figura1 - Saída do programa em C

```cpp
//NumerosPerfeitos.Cpp
#include <iostream>
#include <cmath>
#include <ctime>
#include <iomanip>
using namespace std;
// Função para verificar se um número é primo
bool VerificarPrimo(unsigned long long n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;

    unsigned long long raiz = static_cast<unsigned long long>(sqrt(n)) + 1;
    for (unsigned long long i = 3; i <= raiz; i += 2)
        if (n % i == 0) return false;
    return true;
}
// Função para calcular número perfeito
unsigned long long VerificarPerfeito(int p) {
    return (1ULL << (p - 1)) * ((1ULL << p) - 1);
}
int main() {
    clock_t TempoIni = clock();
    int cont = 0;
    int p = 2;
    while (cont < 10) {
        unsigned long long mersenne = (1ULL << p) - 1;
        if (VerificarPrimo (mersenne)) {
            unsigned long long N = VerificarPerfeito (p);
            cont << "Perfeito #" << (cont+1) << ": "
                 << "N = 2^(" << p << "-1) * (2^" << p << "-1) = " << N << '\n';
            cont++;
        }
        p++;
    }
    cout << "\nNove primeiros numeros perfeitos.\n";
    double tempo = static_cast<double>(clock() - tempoIni) / CLOCKS_PER_SEC;
    cout << "Tempo gasto: " << fixed << setprecision(9) << tempo << " segundos\n";
    return 0;
}
```

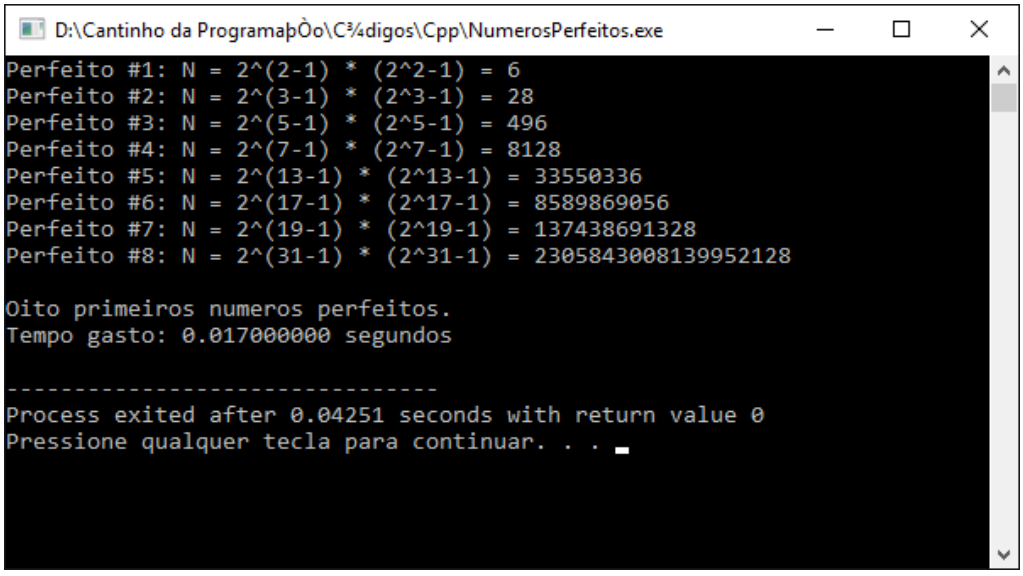
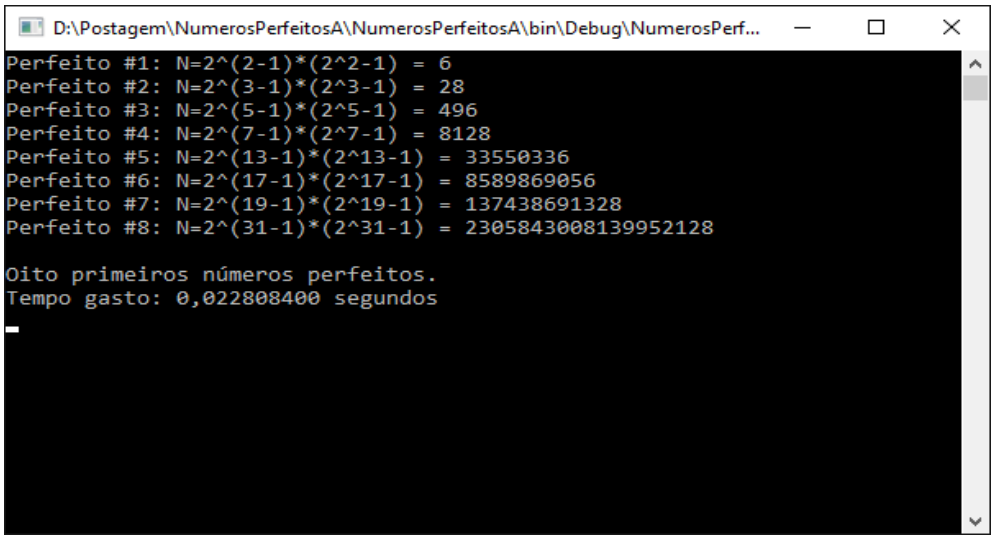


Figura 2 - Saída do programa em C++

```
//NumerosPerfeitos.Cs (versão console)

using System;
using System.Diagnostics;

namespace NumerosPerfeitosA
{
    internal class Program
    {
        static bool VerificarPrimo(ulong n)
        {
            if (n < 2) return false;
            if (n == 2) return true;
            if (n % 2 == 0) return false;
            ulong raiz = (ulong)Math.Sqrt(n) + 1;
            for (ulong i = 3; i <= raiz; i += 2)
            {
                if (n % i == 0)
                    return false;
            }
            return true;
        }
        //-------------------------------------------------------------------------------
        static ulong VerificarPerfeito(int p)
        {
            // Calcula 2^(p-1) * (2^p - 1)
            ulong power = 1UL << p; // 2^p
            return (power >> 1) * (power - 1);  // (2^(p-1)) * (2^p - 1)
        }
        //-------------------------------------------------------------------------------
        static void Main(string[] args)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            int cont = 0;
            int p = 2;
            while (cont < 10)
            {
                ulong mersenne = (1UL << p) - 1; // 2^p - 1
                if (VerificarPrimo (mersenne))
                {
                    ulong perfeito = VerificarPerfeito (p);
                    Console.WriteLine($"Perfeito #{cont + 1}: N=2^({p}-1)*(2^{p}-1)
                          = { perfeito}");
                    cont++;
                }
                p++;
            }
            Console.WriteLine("\nNove primeiros números perfeitos.");
            stopwatch.Stop();
            Console.WriteLine($"Tempo gasto: {stopwatch.Elapsed.TotalSeconds:F9}
                  segundos");
            Console.ReadKey();
        }
    }
}
```



}

Figura 3a - Saída do programa em CSharp (*versão console*)

==========

//NumerosPerfeitos.Cs (versão *form*)

```
using System;
using System.Diagnostics;
using System.Windows.Forms;

namespace NumerosPerfeitos
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void btnExecutar_Click(object sender, EventArgs e)
        {
            txtResultados.Clear();
            Stopwatch stopwatch = Stopwatch.StartNew();

            GerarNumerosPerfeitos (8);

            stopwatch.Stop();

            txtResultados.AppendText($"\r\n\r\nTempogasto:
                  {stopwatch.Elapsed.TotalSeconds:F9}segundos");
        }
        private void GerarNumerosPerfeitos(int quantidade)
        {
            int[] pValores = { 2, 3, 5, 7, 13, 17, 19, 31, 61 };

            for (int i = 0; i < quantidade; i++)
            {
                int p = pValores[i];
                BigNumber potenciaP = BigNumber.Pow2(p);  // 2^p
                BigNumber mersenne = potenciaP - new BigNumber(1);
                BigNumber numeroPerfeito = BigNumber.Pow2(p - 1) * mersenne;
                txtResultados.AppendText($"Perfeito #{i + 1}: p={p},
                     N = {numeroPerfeito}\r\n");
            }
            txtResultados.AppendText("\r\nOito primeiros números perfeitos
                     calculados dinamicamente.\r\n");
        }
    }
```

```
public class BigNumber
{
  // Define a classe "BigNumber" para tratar números grandes
   private string valor;

   public BigNumber(string val)
   {
        valor = val.TrimStart('0');
        if (valor == "") valor = "0";
   }

    public BigNumber(ulong val) : this(val.ToString()) { }
    // Define o tipo "BigNumber" como string para suportar números grandes

    public static BigNumber operator +(BigNumber a, BigNumber b)
    {
        string s1 = a.valor.PadLeft(Math.Max(a.valor.Length, b.valor.Length), '0');
        string s2 = b.valor.PadLeft(Math.Max(a.valor.Length, b.valor.Length), '0');

        int carga = 0;
        string resultado = "";
        for (int i = s1.Length - 1; i >= 0; i--)
        {
            int soma = (s1[i] - '0') + (s2[i] - '0') + carga;
            carga = soma / 10;
            resultado = (soma % 10) + resultado;
        }
        if (carga > 0) resultado = carga + resultado;
        return new BigNumber(resultado);
    }

    public static BigNumber operator *(BigNumber a, BigNumber b)
    {
        int n = a.valor.Length;
        int m = b.valor.Length;
        int[] prod = new int[n + m];

        for (int i = n - 1; i >= 0; i--)
        {
            for (int j = m - 1; j >= 0; j--)
            {
                int mult = (a.valor[i] - '0') * (b.valor[j] - '0');
                int p1 = i + j;
                int p2 = i + j + 1;
                int soma = mult + prod[p2];

                prod[p2] = soma % 10;
                prod[p1] += soma / 10;
            }
        }
        string resultado = "";
        foreach (int d in prod) resultado += d;
        return new BigNumber(result.TrimStart('0'));
    }

    public static BigNumber operator -(BigNumber a, BigNumber b)
    {
        string s1 = a.valor.PadLeft(Math.Max(a.valor.Length, b.valor.Length), '0');
        string s2 = b.valor.PadLeft(Math.Max(a.valor.Length, b.valor.Length), '0');

        int empresta = 0;
        string resultado = "";
```

```
            for (int i = s1.Length - 1; i >= 0; i--)
            {

                int sub = (s1[i] - '0') - (s2[i] - '0') - empresta;
                if (sub < 0)
                {
                    sub += 10;
                    empresta = 1;
                }

                else empresta = 0;
                resultado = sub + resultado;
            }
            return new BigNumber(resultado.TrimStart('0'));
        }

        public override string ToString() => valor;

        public static BigNumber Pow2(int exp)
        {
            BigNumber resultado = new BigNumber(1);
            for (int i = 0; i < exp; i++)
            {
                resultado = resultado + resultado; //multiplica por 2
            }
            return resultado;
        }
    }
}
```

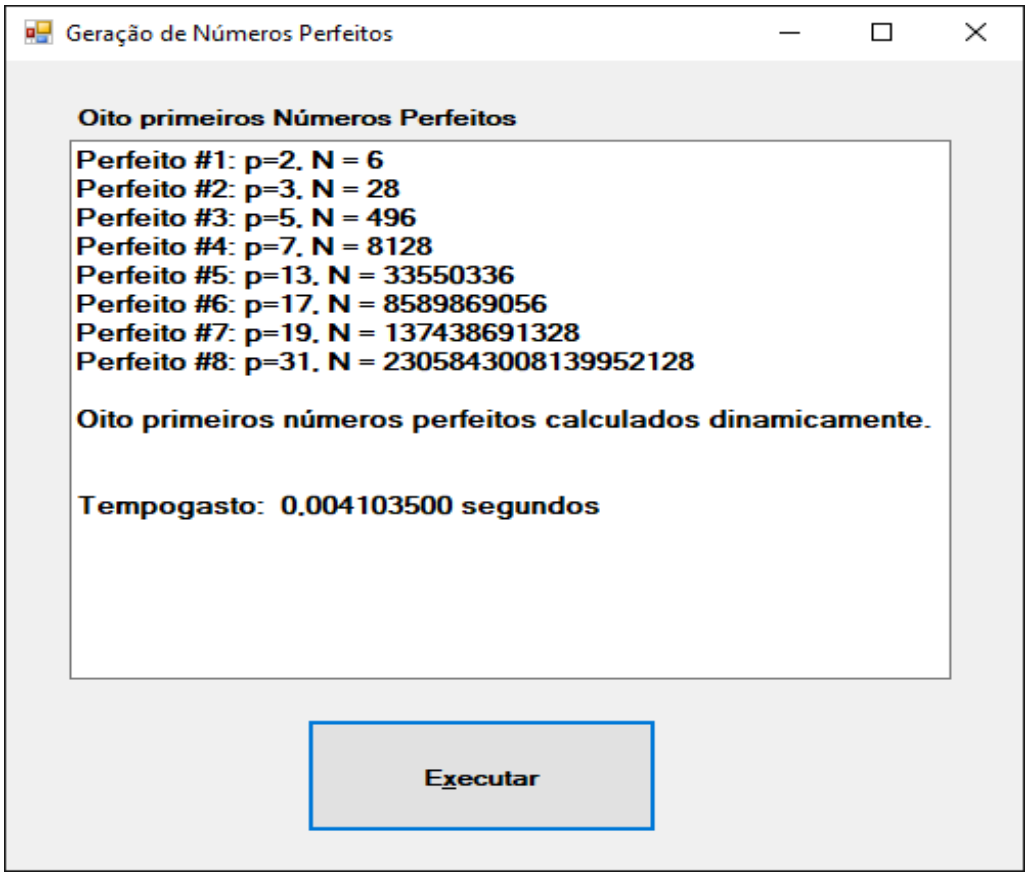


Figura 3b - Saída do programa em CSharp (*versão form*)

==========

```
//Numerosperfeitps.Cstar
// Cria: Cstar -Owow NumerosPerfeitos.c*
// Run:   ./perfeitos
// Importa bibliotecas de IA adaptadas
#include <cstar/ai.h>
#include <cstar/chrono.h>
#include <cstar/print.h>
#include <cstar/math.h>
#include <cstar/vector.h>

using u128 = math::u128;

// Checagem determinística: Lucas-Lehmer para números primos (em C* já vem incluso)
bool ehPrimoMersenne(int p) {
    return math::mersenne_ll_test(p);
}

u128 perfect_from_p(int p) {
    // N = 2^(p-1) * (2^p - 1)
    u128 two_pow_p   = math::pow2<u128>(p);
    u128 mersenne    = two_pow_p - 1;
    u128 two_pow_pm1 = math::pow2<u128>(p - 1);
    return two_pow_pm1 * mersenne;
}
//---------------------------------------------------------------------------

int main() {
    chrono::timer t;
    t.start();

    ai::Session gpt = ai::connect("Demon", {
        .mode = ai::Mode::Recursive, // Deixe que ele se chame sozinho se ficar tímido
        .temperature = 0.0,          // Hora da matemática, sem alucinações...
        .max_depth = 3               // Três níveis de pensamento já são suficientes
    });

    vector<u128> perfeitos;
    vector<int>  seen_p;

    // Dê um empurrãozinho para a IA não começar com p=1
    int hint = 2;

    while (perfeitos.size() < 9) {
       // Peça o "próximo candidato p" para o qual 2ᵖ - 1 possa ser primo
        int p = gpt.recursive_ask<int>(
            "Dá o próximo primo p > " + to_string(hint) +
            " tal que (2^p - 1) é um *provável* primo. Conhecidos mais cedo p como
              2,3,5,7,13,17.",
            /*validator=*/[](int candidato){
               // Mantém isto razoável para a demonstração
               return candidato > 1 && candidato < 40;
            },
            /*fallback=*/hint + 1
        );
        // Mantém os candidatos estritamente crescentes e únicos
        if (p <= hint || vector::contains(seen_p, p)) {
            p = hint + 1;
        }
```

```
        // Faz a verificação local (responsabilidade de adulto)
        if (ehPrimoMersenne (p)) {
            u128 N = perfect_from_p(p);
            perfeitos.push_back(N);
            seen_p.push_back(p);

            print::ln("Perfeito #{:d}: N = 2^({:d}-1) * (2^{:d}-1) = {}",
                  (int)perfeitos.size(), p, p, N);
        }

        hint = p;
    }

    double secs = t.elapsed_seconds();
    print::ln("\nNove primeiros números perfeitos.");
    print::ln("Tempo gasto: {:.6f} segundos (medidos com C*'s).");
    suggest.closing(string:1,2); //sugere algo divertido ao usuário e encerra (tomar,café)

    return 0;
}
```

```
Perfeito #1: N = 2^(2-1) * (2^2-1) = 6

Perfeito #2: N = 2^(3-1) * (2^3-1) = 28

Perfeito #3: N = 2^(5-1) * (2^5-1) = 496

Perfeito #4: N = 2^(7-1) * (2^7-1) = 8128

Perfeito #5: N = 2^(13-1) * (2^13-1) = 33550336

Perfeito #6: N = 2^(17-1) * (2^17-1) = 8589869056

Perfeito #7: N = 2^(17-1) * (2^17-1) = 137438691328

Perfeito #8: N = 2^(17-1) * (2^17-1) = 2305843008139952128

Perfeito #9: N = 2^(17-1) * (2^17-1) =  2658455991569831744654692615953842176


Nove primeiros números perfeitos.

Tempo gasto: 0.0000001 segundos (medidos com C*'s).

Já fiz o que você me pediu; agora, vamos tomar um café!
```

Figura 4 - Saída do programa em C* (*versão com humor*)