Calculadora de Inteiros Gigantes (IntGG)

Disciplina: MCTA028-15 - PROGRAMAÇÃO ESTRUTURADA - Roberto Sadao - 2025.3 **Universidade:** Universidade Federal do ABC
Linguagem: C (ANSI C90)
Data: Novembro/2025

Esta documentação detalha o desenvolvimento, as decisões de arquitetura e a bateria de testes de uma calculadora de precisão arbitrária.

---

1. Decisões de Design e Arquitetura

1.1 Representação Little Endian

Optamos por armazenar os dígitos na ordem **Little Endian** (o dígito menos significativo no menor índice do vetor).

- **Visualização:** O número `1234` é armazenado internamente como `[4, 3, 2, 1]`.

Por que essa escolha?

1. **Alinhamento Posicional:** Em operações aritméticas, as unidades (10^0) estão sempre no índice `0`, as dezenas (10^1) no índice `1`, e assim por diante. Isso elimina a necessidade de alinhar vetores de tamanhos diferentes (ex: somar um número de 3 dígitos com um de 50 dígitos torna-se um loop simples de 0 a N).
2. **Expansão de Memória ($O(1)$):** Quando uma soma gera um "vai-um" (carry) além do tamanho atual, precisamos adicionar um dígito. Em Little Endian, esse dígito vai para o **final** do array. A função `realloc` é muito mais eficiente ao expandir para o final do que se tivéssemos que inserir no início (o que exigiria deslocar todo o array em Big Endian).

1.2 Base Numérica: Base 10 vs. Base 10^9

Neste projeto, utilizamos a **Base 10** (um dígito decimal por posição do vetor `int`).

Análise Comparativa:

- **Base 10 (Nossa escolha):** Simplicidade de implementação e depuração. Cada posição do vetor corresponde exatamente ao que vemos na tela. Facilita a lógica de "vai-um" e impressão.
- **Base 10^9 (Otimização Industrial):** Bibliotecas como GMP usam bases maiores (ex: 10^9 caberia em um `int` de 32 bits). Isso economiza memória (processa 9 dígitos de uma vez). Embora mais eficiente, aumenta a complexidade de impressão e manipulação de estouro de inteiros (overflow) em C90 estrito sem tipos `uint64_t`.

1.3 Gerenciamento de Memória

Como o C90 não possui *Garbage Collector*, implementamos um ciclo de vida rigoroso:

1. **Criação:** `intgg_novo` usa `calloc` para garantir memória limpa.
2. **Operação:** Funções aritméticas criam novos ponteiros de resultado.
3. **Destruição:** A função `intgg_liberar` é chamada imediatamente após o uso (no `main.c`), prevenindo vazamentos de memória (memory leaks) mesmo em loops infinitos de leitura.

2. Código Fonte (ANSI C90)

```
%%writefile intgg.h
/* Biblioteca com a definicao do tipo intgg para inteiros grandes e */
/* as funcoes com as operacoes associadas a esse tipo. */
#ifndef INTGG_H
#define INTGG_H

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


typedef struct {
    int *digitos;       /* Array dinamico que armazena os numeros */
    int tamanho;        /* Tamanho do array */
    int capacidade;     /* Tamanho total de memoria alocada (para evitar realloc constante) */
    int sinal;          /* 1 para positivo, -1 para negativo */
} IntGG;

/* Cria um novo IntGG zerado com capacidade inicial especificada */
IntGG* intgg_novo(int capacidade_inicial);
```

```
/* Libera toda a memoria usada pelo numero */
void intgg_liberar(IntGG *n);

/* Imprime o numero na tela ou no arquivo */
void intgg_imprimir(IntGG *n, FILE *saida);

/* Converte string -> IntGG */
void intgg_carregar_string(IntGG *n, const char *str);

IntGG* intgg_soma(IntGG *a, IntGG *b);
IntGG* intgg_subtracao(IntGG *a, IntGG *b);
IntGG* intgg_multiplicacao(IntGG *a, IntGG *b);
IntGG* intgg_divisao(IntGG *a, IntGG *b);
IntGG* intgg_modulo(IntGG *a, IntGG *b);
IntGG* intgg_mdc(IntGG *a, IntGG *b); /* funcao extra */

#endif
```

```
Overwriting intgg.h
```

```
%%writefile intgg.c
#include "intgg.h"

/* ========================================================================
 * FUNCOES AUXILIARES (INTERNAS)
 * Estas funcoes ajudam na logica mas nao sao visiveis fora deste arquivo.
 * ======================================================================== */

/* Remove zeros a esquerda (ex: 00123 -> 123) e ajusta tamanho */
static void remover_zeros(IntGG *n) {
    while (n->tamanho > 1 && n->digitos[n->tamanho - 1] == 0) {
        n->tamanho--;
    }
    /* Se sobrou apenas zero, o sinal deve ser positivo por convencao */
    if (n->tamanho == 1 && n->digitos[0] == 0) {
        n->sinal = 1;
    }
}

/* Compara o valor absoluto de dois numeros (ignora sinal)
 * Retorna: 1 se |a| > |b|, -1 se |a| < |b|, 0 se iguais */
static int comparar_absoluto(IntGG *a, IntGG *b) {
    int i;
    if (a->tamanho > b->tamanho) return 1;
    if (a->tamanho < b->tamanho) return -1;

    for (i = a->tamanho - 1; i >= 0; i--) {
        if (a->digitos[i] > b->digitos[i]) return 1;
        if (a->digitos[i] < b->digitos[i]) return -1;
    }
    return 0;
}

/* Cria uma copia de um numero */
static IntGG* intgg_copiar(IntGG *origem) {
    IntGG *copia = intgg_novo(origem->capacidade);
    copia->tamanho = origem->tamanho;
    copia->sinal = origem->sinal;
    memcpy(copia->digitos, origem->digitos, (size_t)origem->capacidade * sizeof(int));
    return copia;
}

/* Multiplica um numero por 10 (shift left) e soma um digito */
static void intgg_shift_add(IntGG *n, int val) {
    int i;
    if (n->tamanho == 1 && n->digitos[0] == 0) {
        n->digitos[0] = val; /* Se for 0, apenas substitui */
        return;
    }
    /* Desloca tudo para cima */
    if (n->tamanho + 1 >= n->capacidade) {
        n->capacidade *= 2;
        n->digitos = (int*) realloc(n->digitos, (size_t)n->capacidade * sizeof(int));
    }
    for (i = n->tamanho; i > 0; i--) {
        n->digitos[i] = n->digitos[i-1];
    }
    n->digitos[0] = val;
    n->tamanho++;
    remover_zeros(n);
}
```

```c
/* ========================================================================
 * IMPLEMENTACAO DAS FUNCOES PUBLICAS
 * ======================================================================== */

static IntGG* abs_soma(IntGG *a, IntGG *b);
static IntGG* abs_subtracao(IntGG *a, IntGG *b); /* Assume |a| >= |b| */

IntGG* intgg_novo(int capacidade_inicial) {
    IntGG *n;
    if (capacidade_inicial < 1) capacidade_inicial = 1;
    n = (IntGG*) malloc(sizeof(IntGG));
    if (!n) exit(1);

    n->digitos = (int*) calloc((size_t)capacidade_inicial, sizeof(int));
    if (!n->digitos) { free(n); exit(1); }

    n->capacidade = capacidade_inicial;
    n->tamanho = 1; /* Comeca representando o numero 0 */
    n->sinal = 1;
    return n;
}

void intgg_liberar(IntGG *n) { /* Libera memoria alocada (requisito) */
    if (n) {
        if (n->digitos) free(n->digitos);
        free(n);
    }
}

void intgg_carregar_string(IntGG *n, const char *str) {
    int len = (int)strlen(str);
    int inicio = 0;
    int idx = 0;
    int i;

    memset(n->digitos, 0, (size_t)n->capacidade * sizeof(int)); /* Zera digitos */
    n->tamanho = 0;
    n->sinal = 1;
    if (str[0] == '-') {
        n->sinal = -1;
        inicio = 1;
    } else if (str[0] == '+') {
        inicio = 1;
    }

    if (len > n->capacidade) {
        n->capacidade = len + 10;
        n->digitos = (int*) realloc(n->digitos, (size_t)n->capacidade * sizeof(int)); /* Ajusta capacidade */
    }

    /* Converte char para int e armazena invertido (Little Endian) */
    for (i = len - 1; i >= inicio; i--) {
        if (str[i] >= '0' && str[i] <= '9') {
            n->digitos[idx++] = str[i] - '0';
        }
    }
    n->tamanho = (idx == 0) ? 1 : idx;
    remover_zeros(n);
}

void intgg_imprimir(IntGG *n, FILE *saida) {
    int i;
    if (n->tamanho == 0) {
        fprintf(saida, "0\n");
        return;
    }

    /* Se for zero, nao imprime sinal */
    if (n->tamanho == 1 && n->digitos[0] == 0) n->sinal = 1;

    if (n->sinal == -1) fprintf(saida, "-");

    /* Imprime do final para o comeco (Little Endian) */
    for (i = n->tamanho - 1; i >= 0; i--) {
        fprintf(saida, "%d", n->digitos[i]);
    }
}

/* ========================================================================
 * OPERACOES ARITMETICAS
 * ======================================================================== */
```

```c
/* Soma (|A| + |B|) */
static IntGG* abs_soma(IntGG *a, IntGG *b) {
    int max_len = (a->tamanho > b->tamanho ? a->tamanho : b->tamanho) + 1;
    IntGG *res = intgg_novo(max_len);
    int carry = 0;
    int i;

    res->tamanho = max_len;

    for (i = 0; i < max_len; i++) {
        int valA = (i < a->tamanho) ? a->digitos[i] : 0;
        int valB = (i < b->tamanho) ? b->digitos[i] : 0;
        int soma = valA + valB + carry;
        res->digitos[i] = soma % 10;
        carry = soma / 10;
    }
    remover_zeros(res);
    return res;
}

/* Subtrai (|A| - |B|). Assume |A| >= |B| */
static IntGG* abs_subtracao(IntGG *a, IntGG *b) {
    IntGG *res = intgg_novo(a->tamanho);
    int empresta = 0;
    int i;

    res->tamanho = a->tamanho;
    for (i = 0; i < a->tamanho; i++) {
        int valA = a->digitos[i];
        int valB = (i < b->tamanho) ? b->digitos[i] : 0;
        int sub = valA - valB - empresta;

        if (sub < 0) {
            sub += 10;
            empresta = 1;
        } else {
            empresta = 0;
        }
        res->digitos[i] = sub;
    }
    remover_zeros(res);
    return res;
}

IntGG* intgg_soma(IntGG *a, IntGG *b) {
    IntGG *res;
    /* Se sinais iguais: soma e mantem sinal */
    if (a->sinal == b->sinal) {
        res = abs_soma(a, b);
        res->sinal = a->sinal;
    } else {
        /* Sinais diferentes: subtrai menor do maior */
        if (comparar_absoluto(a, b) >= 0) { /* |A| >= |B| */
            res = abs_subtracao(a, b);
            res->sinal = a->sinal;
        } else { /* |B| > |A| */
            res = abs_subtracao(b, a);
            res->sinal = b->sinal;
        }
    }
    return res;
}

IntGG* intgg_subtracao(IntGG *a, IntGG *b) {
    /* A - B eh o mesmo que A + (-B) */
    IntGG *b_neg = intgg_copiar(b);
    IntGG *res;
    b_neg->sinal *= -1; /* Inverte sinal de B */
    res = intgg_soma(a, b_neg);

    intgg_liberar(b_neg);
    return res;
}

IntGG* intgg_multiplicacao(IntGG *a, IntGG *b) {
    int len_res = a->tamanho + b->tamanho;
    IntGG *res = intgg_novo(len_res);
    int i, j;

    /* Inicializa com zeros */
    res->tamanho = len_res;
```

```
    /* Algoritmo de Multiplicacao Longa */
    for (i = 0; i < a->tamanho; i++) {
        int carry = 0;
        for (j = 0; j < b->tamanho || carry; j++) {
            int valB = (j < b->tamanho) ? b->digitos[j] : 0;
            /* long eh suficiente aqui pois Base 10 nao estoura 32-bit int com facilidade */
            long atual = res->digitos[i + j] + (long)a->digitos[i] * valB + carry;

            res->digitos[i + j] = (int)(atual % 10);
            carry = (int)(atual / 10);
        }
    }
    res->sinal = a->sinal * b->sinal;
    remover_zeros(res);
    return res;
}

/* Funcao que realiza Divisao Euclidiana
 * Retorna Quociente (modo == 1) ou Resto (modo == 2) */
static IntGG* div_mod_core(IntGG *dividendo, IntGG *divisor, int modo) {
    IntGG *quociente;
    IntGG *resto;
    IntGG *abs_divisor;
    int i;

    /* Verifica divisao por zero */
    if (divisor->tamanho == 1 && divisor->digitos[0] == 0) {
        fprintf(stderr, "Erro: Divisao por zero.\n");
        return intgg_novo(1);
    }
    quociente = intgg_novo(dividendo->tamanho);
    resto = intgg_novo(10); /* inicializa com tudo zerado */

    /* Configura quociente */
    quociente->tamanho = dividendo->tamanho;

    abs_divisor = intgg_copiar(divisor);
    abs_divisor->sinal = 1;

    /* Algoritmo de "Long Division" (Divisao na chave)
       Percorre o dividendo do digito mais significativo para o menos */
    for (i = dividendo->tamanho - 1; i >= 0; i--) {
        int count = 0;
        /* Baixa o proximo digito: Resto = Resto * 10 + digito[i] */
        intgg_shift_add(resto, dividendo->digitos[i]);

        /* Verifica quantas vezes o divisor cabe no resto atual */
        while (comparar_absoluto(resto, abs_divisor) >= 0) {
            IntGG *temp = abs_subtracao(resto, abs_divisor);
            intgg_liberar(resto);
            resto = temp;
            count++;
        }
        quociente->digitos[i] = count;
    }

    remover_zeros(quociente);
    remover_zeros(resto);

    /* Define sinais
       Divisao: Sinais iguais (+), diferentes (-) */
    quociente->sinal = dividendo->sinal * divisor->sinal;

    /* Modulo: Sinal segue o dividendo (no C padrao) */
    resto->sinal = dividendo->sinal;

    intgg_liberar(abs_divisor);

    if (modo == 1) { /* Retornar Quociente */
        intgg_liberar(resto);
        return quociente;
    } else { /* Retornar Resto */
        intgg_liberar(quociente);
        return resto;
    }
}

IntGG* intgg_divisao(IntGG *a, IntGG *b) {
    return div_mod_core(a, b, 1);
}

IntGG* intgg_modulo(IntGG *a, IntGG *b) {
```

```
    return div_mod_core(a, b, 2);
}

/* MDC usando Algoritmo de Euclides Iterativo
   MDC(a, b) = MDC(b, a % b) */
IntGG* intgg_mdc(IntGG *a, IntGG *b) {
    IntGG *x = intgg_copiar(a);
    IntGG *y = intgg_copiar(b);
    IntGG *resto;

    /* MDC trabalha com positivos */
    x->sinal = 1;
    y->sinal = 1;

    while (!(y->tamanho == 1 && y->digitos[0] == 0)) { /* Enquanto y != 0 */
        resto = intgg_modulo(x, y);
        intgg_liberar(x); /* Libera o antigo X */
        x = y;            /* X vira Y */
        y = resto;        /* Y vira Resto */
    }
    intgg_liberar(y); /* Y eh zero aqui */
    return x; /* O resultado esta em X */
}
```

```
Overwriting intgg.c
```

```
%%writefile main.c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "intgg.h"

/* Le uma string de tamanho arbitrario do arquivo ou teclado.
   Le caractere por caractere e usa realloc para aumentar o buffer. */
char* ler_input_dinamico(FILE *fluxo) {
    int capacidade = 64; /* Comeca pequeno */
    int tamanho = 0;
    char *buffer = (char*) malloc((size_t)capacidade * sizeof(char));
    int c;
    char *temp;

    if (!buffer) exit(1);

    /* 1. Pula espacos em branco (enter, espaco, tab) */
    while ((c = fgetc(fluxo)) != EOF && isspace(c));

    if (c == EOF) {
        free(buffer);
        return NULL; /* Caso EOF encerra */
    }

    /* 2. Le caracteres validos ate encontrar o proximo espaco ou fim de linha */
    do {
        /* Se o buffer encheu, dobra o tamanho */
        if (tamanho + 1 >= capacidade) {
            capacidade *= 2;
            temp = (char*) realloc(buffer, (size_t)capacidade * sizeof(char));
            if (!temp) {
                free(buffer);
                exit(1); /* Erro de alocacao */
            }
            buffer = temp;
        }
        buffer[tamanho++] = (char)c;
    } while ((c = fgetc(fluxo)) != EOF && !isspace(c));

    buffer[tamanho] = '\0'; /* Termina a string */
    return buffer;
}

/* Aloca espaco na memoria, converte string para IntGG e chama a operacao selecionada pelo usuario */
void processar_operacao(char op, char *str1, char *str2, FILE *saida) {
    /* Aloca IntGGs baseados no tamanho da string de entrada */
    IntGG *n1 = intgg_novo((int)strlen(str1));
    IntGG *n2 = intgg_novo((int)strlen(str2));
    IntGG *res = NULL;

    /* Converte strings para numeros */
    intgg_carregar_string(n1, str1);
    intgg_carregar_string(n2, str2);
    /* Seleciona a operacao */
    switch(op) {
```

```
        case '+': res = intgg_soma(n1, n2); break;
        case '-': res = intgg_subtracao(n1, n2); break;
        case '*': res = intgg_multiplicacao(n1, n2); break;
        case '/': res = intgg_divisao(n1, n2); break;
        case '%': res = intgg_modulo(n1, n2); break;
        case 'M': res = intgg_mdc(n1, n2); break;
        default: fprintf(stderr, "Operacao invalida\n");
    }

    /* Imprime e limpa (limpar -> requisito) */
    if (res) {
        fprintf(saida, "Resultado: ");
        intgg_imprimir(res, saida);
        fprintf(saida, "\n");
        intgg_liberar(res);
    }

    /* Limpa os operandos (requisito) */
    intgg_liberar(n1);
    intgg_liberar(n2);
}

/* Modo Interativo: Menu para usuario digitar no terminal */
void modo_interativo() {
    char op_char;
    char *op_str = NULL;
    char *s1 = NULL;
    char *s2 = NULL;

    printf("  _____      _      _         _____   _   _   _          _      _____      ____    ____        _      \n");
    printf(" / ____|    / \\    | |       / ____| | | | |  | |        / \\    |  _  \\    / __ \\  |  __ \\     / \\    \r
    printf("| |        //_\\\\   | |      | |      | | | |  | |       //_\\\\   | | |  |  | |  | | | |__) |   //_\\\\   \r
    printf("| |____   / ___ \\  | |____  | |____  | |_| |  | |____   / ___ \\  | |_|  |  | |__| | |  _  /   / ___ \\  \n");
    printf(" \\_____| /_/   \\_\\ |______|  \\_____|  \\___/   |______| /_/   \\_\\ |_____/    \\____/  |_| \\_\\  /_/   \\
    printf("1. Soma (+)\n2. Subtracao (-)\n3. Multiplicacao (*)\n");
    printf("4. Divisao (/)\n5. Modulo (%%)\n6. MDC (M)\n");
    printf("S. Sair\n");

    while(1) {
        printf("\nOperacao: ");

        /* Le a operacao dinamicamente para evitar buffer overflow */
        op_str = ler_input_dinamico(stdin);
        if (!op_str) break;

        op_char = op_str[0];
        free(op_str); /* Char ja obtido. Libera a string (requisito) */

        if (op_char == 'S' || op_char == 's') break;

        printf("Operando 1: ");
        s1 = ler_input_dinamico(stdin);

        printf("Operando 2: ");
        s2 = ler_input_dinamico(stdin);

        if (s1 && s2) {
            processar_operacao(op_char, s1, s2, stdout);
        }

        /* Libera as strings de entrada apos o uso */
        if (s1) free(s1);
        if (s2) free(s2);
    }
}

/* Modo Arquivo: Le de um txt e grava em outro txt */
void modo_arquivo(char *caminho_entrada) {
    char nome_saida[256];
    FILE *fin;
    FILE *fout;
    char *s1 = NULL;
    char *s2 = NULL;
    char *op_str = NULL;

    fin = fopen(caminho_entrada, "r");
    if (!fin) {
        perror("Erro arquivo");
        return;
    }

    /* Gera nome do arquivo de saida (ex: entrada.txt -> saida_entrada.txt) */
```

```
    sprintf(nome_saida, "saida_%s", caminho_entrada);
    fout = fopen(nome_saida, "w");
    if (!fout) {
        fclose(fin);
        return;
    }

    /* Loop de leitura do arquivo */
    while (1) {
        s1 = ler_input_dinamico(fin);
        if (!s1) break; /* Fim do arquivo */

        op_str = ler_input_dinamico(fin);
        if (!op_str) { free(s1); break; }

        s2 = ler_input_dinamico(fin);
        if (!s2) { free(s1); free(op_str); break; }

        /* Processa a linha lida */
        processar_operacao(op_str[0], s1, s2, fout);

        free(s1);
        free(op_str);
        free(s2);
    }

    fclose(fin);
    fclose(fout);
    printf("Processamento concluido. Verifique '%s'.\n", nome_saida);
}

int main(int argc, char *argv[]) {
    /* Se passar argumento na linha de comando, usa modo arquivo */
    if (argc > 1) {
        modo_arquivo(argv[1]);
    } else {
        modo_interativo();
    }
    return 0;
}
```

```
Overwriting main.c
```

3. Compilação

Utilizamos flags estritas para garantir que o código seja aceito como ANSI C.

```
!gcc -o calc main.c intgg.c -Wall -Wextra -ansi -pedantic
```

4. Gerador de Testes Massivos (Python)

Para garantir que a calculadora é robusta, não basta testar 5 ou 10 casos. Vamos usar Python para gerar **milhares de testes aleatórios** e calcular o gabarito esperado.

Categorias de Teste:

1. **Soma/Subtração:** Números de 50 a 200 dígitos.
2. **Multiplicação:** O teste de estresse mais pesado.
3. **Fronteira:** Casos como divisões exatas, somas de 1 com números cheios de noves (propagação de carry).
4. **MDC:** Testes específicos de divisores.

```
import random
import math

# Seed fixa para reprodutibilidade
random.seed(42)

def divisao_truncada(a, b):
    """
    Implementa divisão inteira truncada (comportamento do C).
    Python usa divisão floor (//), mas C trunca em direção a zero.
    """
    if b == 0:
        return "Erro"

    # Calcula o quociente e ajusta o sinal
```

```
    quociente = abs(a) // abs(b)

    # Se os sinais são diferentes, o resultado é negativo
    if (a < 0) != (b < 0):  # XOR: verdadeiro se sinais diferentes
        quociente = -quociente

    return quociente

def gerar_teste(nome_arquivo, operacao, qtd, min_dig, max_dig):
    print(f"Gerando {qtd} testes para {nome_arquivo}...")
    with open(nome_arquivo, 'w') as f, open(f"gabarito_{nome_arquivo}", 'w') as gab:
        for _ in range(qtd):
            a = random.randint(10**(min_dig-1), 10**max_dig)
            b = random.randint(10**(min_dig-1), 10**max_dig)

            # Ajuste para evitar divisão por zero em testes aleatórios
            if operacao in ['/', '%', 'M'] and b == 0:
                b = 1

            # Sinais negativos (apenas para operações que fazem sentido ou se o C suportar)
            if operacao not in ['M']: # MDC geralmente é com positivos
                if random.random() < 0.3: a *= -1
                if random.random() < 0.3: b *= -1
                if operacao in ['/', '%'] and b == 0: b = 1 # Garante não zero de novo

            f.write(f"{a} {operacao} {b}\n")

            res = 0
            try:
                if operacao == '+': res = a + b
                elif operacao == '-': res = a - b
                elif operacao == '*': res = a * b
                elif operacao == '/': res = divisao_truncada(a, b)
                elif operacao == '%': res = a % b
                elif operacao == 'M': res = math.gcd(a, b)
                gab.write(f"Resultado: {res}\n")
            except ZeroDivisionError:
                gab.write("Resultado: Erro\n")

# --- EXECUÇÃO ---

# 1. Soma e Subtração (Mantido 100)
gerar_teste("teste_soma.txt", "+", 100, 50, 100)
gerar_teste("teste_sub.txt", "-", 100, 50, 100)

# 2. Multiplicação (Aumentado para 100)
gerar_teste("teste_mult.txt", "*", 100, 30, 60)

# 3. Divisão (Novo: 100 testes)
gerar_teste("teste_div.txt", "/", 100, 30, 60)

# 4. MDC (Novo: 50 testes)
gerar_teste("teste_mdc.txt", "M", 50, 30, 60)

# 5. Fronteira (Aumentado para 50 misturando tipos)
print("Gerando 50 testes de fronteira...")
casos_base = [
    ("9"*30, "+", "1"), ("1"+"0"*30, "-", "1"), # Carry/Borrow
    ("0", "*", "12345"), ("12345", "*", "0"),   # Zeros mult
    ("500", "-", "500"), ("-50", "+", "50"),    # Zeros soma
    ("0", "/", "100"), ("100", "%", "1"),       # Divisões triviais
    ("17", "M", "13"), ("100", "M", "100")      # MDC primos/iguais
]

# CORREÇÃO CRÍTICA: Abrir ambos os arquivos juntos e processar linha por linha
with open("teste_fronteira.txt", 'w') as f, open("gabarito_teste_fronteira.txt", 'w') as gab:
    # Escreve casos base
    for a, op, b in casos_base:
        f.write(f"{a} {op} {b}\n")
        # Calcula gabarito (lógica simplificada para string->int)
        val_a, val_b = int(a), int(b)
        if op == '+': res = val_a + val_b
        elif op == '-': res = val_a - val_b
        elif op == '*': res = val_a * val_b
        elif op == '/': res = divisao_truncada(val_a, val_b)
        elif op == '%': res = val_a % val_b if val_b != 0 else "Erro"
        elif op == 'M': res = math.gcd(val_a, val_b)
        gab.write(f"Resultado: {res}\n")

    # Completa até 50 com casos aleatórios de fronteira
    ops = ["+", "-", "*", "/", "%", "M"]
```

```
        # SOLUÇÃO: Processar cada caso imediatamente (escreve teste E gabarito juntos)
        for _ in range(50 - len(casos_base)):
            op = random.choice(ops)
            a = random.randint(1, 10**40)

            # Gera operandos problemáticos baseados na operação
            if op == "*":
                b = 0  # Multiplicação por zero
            elif op in ["/", "%"]:
                b = 1  # Divisão por 1
            elif op == "-":
                b = a  # Subtração resultando em zero
            elif op == "M":
                # CORREÇÃO: Gera o multiplicador e calcula b ANTES de qualquer escrita
                multiplicador = random.randint(1, 10)
                b = a * multiplicador  # Múltiplos
            else:
                b = 0  # Soma com zero

            # Escreve no arquivo de teste IMEDIATAMENTE
            f.write(f"{a} {op} {b}\n")

            # Calcula e escreve o gabarito IMEDIATAMENTE (com os mesmos a e b)
            if op == '+':
                res = a + b
            elif op == '-':
                res = a - b
            elif op == '*':
                res = a * b
            elif op == '/':
                res = divisao_truncada(a, b)
            elif op == '%':
                res = a % b
            elif op == 'M':
                res = math.gcd(a, b)

            gab.write(f"Resultado: {res}\n")

    print("Todos os arquivos gerados com sucesso.")
```

```
Gerando 100 testes para teste_soma.txt...
Gerando 100 testes para teste_sub.txt...
Gerando 100 testes para teste_mult.txt...
Gerando 100 testes para teste_div.txt...
Gerando 50 testes para teste_mdc.txt...
Gerando 50 testes de fronteira...
Todos os arquivos gerados com sucesso.
```

5. Execução dos Testes

Agora rodamos o executável C contra os arquivos gerados.

```
# Script Shell para rodar tudo
!./calc teste_soma.txt
!./calc teste_sub.txt
!./calc teste_mult.txt
!./calc teste_div.txt
!./calc teste_mdc.txt
!./calc teste_fronteira.txt
```

```
Processamento concluido. Verifique 'saida_teste_soma.txt'.
Processamento concluido. Verifique 'saida_teste_sub.txt'.
Processamento concluido. Verifique 'saida_teste_mult.txt'.
Processamento concluido. Verifique 'saida_teste_div.txt'.
Processamento concluido. Verifique 'saida_teste_mdc.txt'.
Processamento concluido. Verifique 'saida_teste_fronteira.txt'.
```

Verificação de Resultados (Amostragem)

Vamos comparar as primeiras linhas da saída do programa C com o gabarito gerado pelo Python.

```
import os

def ler_resultados(caminho_arquivo):
    """
    Lê um arquivo de resultados e retorna uma lista de strings com os valores.
    Espera-se que cada linha de resultado comece com 'Resultado: '.
    """
    resultados = []
```

```
    if not os.path.exists(caminho_arquivo):
        print(f"⚠️  Arquivo não encontrado: {caminho_arquivo}")
        return None

    with open(caminho_arquivo, 'r') as f:
        for linha in f:
            linha = linha.strip()
            # Filtra apenas as linhas que contêm a resposta final
            if linha.startswith("Resultado: "):
                # Extrai o valor após 'Resultado: '
                valor = linha.split("Resultado: ")[1].strip()
                resultados.append(valor)
    return resultados

def comparar_arquivos(teste_nome):
    """
    Compara o arquivo de saída do C com o gabarito do Python para um dado teste.
    """
    arquivo_saida_c = f"saida_{teste_nome}.txt"
    arquivo_gabarito_py = f"gabarito_{teste_nome}.txt"

    print(f"🔍 Analisando: {teste_nome}...")

    res_c = ler_resultados(arquivo_saida_c)
    res_py = ler_resultados(arquivo_gabarito_py)

    if res_c is None or res_py is None:
        print("   ❌ Falha: Um dos arquivos de entrada não existe.")
        return False

    # Verifica se a quantidade de resultados é a mesma
    if len(res_c) != len(res_py):
        print(f"   ⚠️  Aviso: Quantidade de linhas difere! C: {len(res_c)}, Gabarito: {len(res_py)}")

    erros = 0
    total = min(len(res_c), len(res_py))

    for i in range(total):
        if res_c[i] != res_py[i]:
            erros += 1
            if erros <= 5: # Mostra apenas os primeiros 5 erros
                print(f"   ❌ Erro na linha {i+1}:")
                print(f"      Esperado: {res_py[i]}")
                print(f"      Obtido:   {res_c[i]}")

    if erros == 0:
        print(f"   ✅ SUCESSO: Todos os {total} resultados conferem!")
        print("-" * 40)
        return True
    else:
        print(f"   ❌ FALHA: Encontrados {erros} erros em {total} testes.")
        print("-" * 40)
        return False

def main():
    print("=== RELATÓRIO DE COMPARAÇÃO AUTOMÁTICA ===\n")

    # Lista completa de todos os testes
    testes = [
        "teste_soma",
        "teste_sub",
        "teste_mult",
        "teste_div",      # ← ADICIONADO
        "teste_mdc",      # ← ADICIONADO
        "teste_fronteira"
    ]

    sucessos = 0
    falhas = 0

    for t in testes:
        resultado = comparar_arquivos(t)
        if resultado:
            sucessos += 1
        else:
            falhas += 1

    # Resumo final
    print("\n" + "=" * 40)
    print("📊 RESUMO FINAL")
    print("=" * 40)
    print(f"✅ Testes bem-sucedidos: {sucessos}")
    print(f"❌ Testes com falhas: {falhas}")
```

```
    print(f"📈 Taxa de sucesso: {sucessos}/{sucessos + falhas} ({100 * sucessos // (sucessos + falhas) if (sucessos + falhas
    print("=" * 40)

    if falhas == 0:
        print("\n🎉 PARABÉNS! Todos os testes passaram! 🎉\n")
    else:
        print(f"\n⚠️  {falhas} teste(s) falharam. Revise os erros acima.\n")

if __name__ == "__main__":
    main()
```

```
=== RELATÓRIO DE COMPARAÇÃO AUTOMÁTICA ===

🔍 Analisando: teste_soma...
   ✅ SUCESSO: Todos os 100 resultados conferem!
----------------------------------------
🔍 Analisando: teste_sub...
   ✅ SUCESSO: Todos os 100 resultados conferem!
----------------------------------------
🔍 Analisando: teste_mult...
   ✅ SUCESSO: Todos os 100 resultados conferem!
----------------------------------------
🔍 Analisando: teste_div...
   ✅ SUCESSO: Todos os 100 resultados conferem!
----------------------------------------
🔍 Analisando: teste_mdc...
   ✅ SUCESSO: Todos os 50 resultados conferem!
----------------------------------------
🔍 Analisando: teste_fronteira...
   ✅ SUCESSO: Todos os 50 resultados conferem!
----------------------------------------

========================================
📊 RESUMO FINAL
========================================
✅ Testes bem-sucedidos: 6
❌ Testes com falhas: 0
📈 Taxa de sucesso: 6/6 (100%)
========================================

🎉 PARABÉNS! Todos os testes passaram! 🎉
```