ЗМІСТ

0.1 Використання GNU GMP бібліотеки із мовою програмування CSharp

Як вже було згадано вище, GNU GMP допомагає виконувати будь які арифметичні операції із довільною точністю, що обмежена лише пам'яттю машини, на якій виконується программа. Для того, щоб використовувати цю бібліотеку в мові програмування CSharp слід встановити до проєкту Nuget пакет із назвою "Math.Gmp.Native.NET".

Бібліотека "Math.Gmp.Native.NET"автоматично завантажує на виконання 32-бітну або 64-бітну бібліотеку GNU MP, яка відповідає поточній архітектурі центрального процесора, тим самим дозволяючи будувати проекти Visual Studio для режимів Any CPU, x86 або x64. Вона базується на розширеній збірці GNU MP, яка автоматично визначає тип поточного процесора і вибирає оптимізацію коду на мові асемблера для даного процесора, тим самим забезпечуючи найкращу продуктивність.

Клас Math.Gmp.Native.gmp_lib містить статичний метод для кожної з функцій GNU MP. Інші типи визначені для відтворення структур і псевдонімів типів GNU MP і мов C, а також мовних конструкцій C.

Заради зручності цю довідку було створено з офіційного посібника GNU MP версії 6.1.2. Вона демонструє, з прикладами, як кожна функція GNU MP викликається в .NET.

Для початку порівняємо код мовою програмування CSharp із кодом на Rust.(рис. 1)

Видно, що для використання GMP треба використати змінні класу mpz_t і задавати певні значення чисел у функцію mpz_pown. Якщо метод відпрацював успішно, то в консолі можна буде побачити повідомлення про коректну роботу. (рис. 2)

Розглянемо для остаточного розуміння, як саме використовувати написану мовою програмування C бібліотеку у поєднанні із CSharp оболонкою для знаходження зворотнього елементу. (рис. 3)

```
private static void Check_mpz_powm()
{
    // Create, initialize, and set the value of base to 2.
    mpz_t @base = new mpz_t();
    gmp_lib.mpz_init_set_ui(@base, 2U);

    // Create, initialize, and set the value of exp to 4.
    mpz_t exp = new mpz_t();
    gmp_lib.mpz_init_set_ui(exp, 4U);

    // Create, initialize, and set the value of mod to 3.
    mpz_t mod = new mpz_t();
    gmp_lib.mpz_init_set_ui(mod, 3U);

    // Create, initialize, and set the value of rop to 0.
    mpz_t rop = new mpz_t();
    gmp_lib.mpz_init(rop);

    // Set rop = base^exp mod mod.
    gmp_lib.mpz_powm(rop, @base, exp, mod);

    // Assert that rop is 1.
    Console.WriteLine(gmp_lib.mpz_get_si(rop) == 1 ? "Result: 1. Method works correctly." : $"Result: not 1. Method does not works c

    // Release unmanaged memory allocated for rop, base, exp, and mod.
    gmp_lib.mpz_clears(rop, @base, exp, mod, null);
}
```

Рисунок 1 – Приклад піднесення до степеня.

```
Result (mpz_powm): 1. Method works correctly.
Result (BigInteger): 1. Method works correctly
```

Рисунок 2 – Приклад для перевірки коректності роботи алгоритму.

```
1 reference
public static void Check_mpz_invert()
{
    // Create, initialize, and set the value of op1 to 3.
    mpz_t op1 = new mpz_t();
    gmp_lib.mpz_init_set_ui(op1, 3U);

    // Create, initialize, and set the value of op2 to 11.
    mpz_t op2 = new mpz_t();
    gmp_lib.mpz_init_set_ui(op2, 11U);

    // Create, initialize, and set the value of rop to 0.
    mpz_t rop = new mpz_t();
    gmp_lib.mpz_init(rop);

    // Set rop to the modular inverse of op1 mod op2, i.e. b, where op1 * b mod op1 = 1.
    gmp_lib.mpz_invert(rop, op1, op2);

    // Assert that rop is 4,
    Console.WriteLine(gmp_lib.mpz_get_si(rop) == 4
        ? "Result: 4. Method works correctly."
        : $"Result: not 4. Method does not works correctly.");

    // Release unmanaged memory allocated for rop, op1, and op2.
    gmp_lib.mpz_clears(rop, op1, op2, null);
}
```

Рисунок 3 – Приклад знаходження зворотнього елементу.

0.2 Використання System.Numerics.BigInteger бібліотеки із мовою програмування CSharp

BigInteger - це тип даних в програмуванні, який дозволяє представляти та виконувати операції з великими цілими числами, які виходять за межі можливостей стандартних числових типів. В інших мовах програмування цей тип може мати різні назви, але концепція залишається однаковою - він дозволяє працювати з числами довільної довжини і не обмежується розміром пам'яті, яку може займати число.

У .NET BigInteger знаходиться в просторі імен System.Numerics і забезпечує можливість виконувати арифметичні операції, порівнювати та взаємодіяти з великими цілими числами, незалежно від їх розміру, забезпечуючи точність та надійність операцій. Це особливо корисно при виконанні обчислень, де великі числа можуть виникати при розв'язанні математичних задач, криптографії, обробці великих даних тощо.

Таким чином, маємо реалізацію піднесення до степеню, використовуючи System.Numerics.BigInteger.ModPow.

```
private static void Check_ModPow()
{
    // Base, exponent, and modulus values
    BigInteger baseValue = 2;
    BigInteger exponent = 4;
    BigInteger modulus = 3;

    // Calculate (base^exponent) % modulus using BigInteger.ModPow method
    BigInteger result = BigInteger.ModPow(baseValue, exponent, modulus);

    // Assert that rop is 1.
    Console.WriteLine(result == 1 ? "Result (BigInteger): 1. Method works correctly." : $"Result (BigInteger)
}
```

Рисунок 4 – Приклад піднесення до степеня BigInteger.

Після виконання такого шматочку коду в результаті побачимо

```
Result (BigInteger): 1. Method works correctly.
```

Рисунок 5 – Результат піднесення до степеня BigInteger.

Таким чином, бачимо, що обидві функції коректно працюють.

0.3 Висновки

Мова програмування CSharp є відмінним вибором для роботи із різними арифметичними операціями, оскільки вона підтримує різні бібліотеки для роботи з великими числами та арифметикою високої точності. Перш за все, вбудований клас System.Numerics.BigInteger надає можливість оперувати дуже великими цілими числами без втрати точності, що робить його ідеальним вибором для обчислень, де великі числа важливі.

Додатково, наявність зовнішніх бібліотек, таких як Math.Gmp.Native або System.Numerics.MPFR, розширює можливості CSharp. Ці бібліотеки забезпечують швидку та ефективну роботу з великими числами, що особливо корисно для вимогливих до продуктивності застосувань. Наприклад, вони можуть бути використані в задачах криптографії, математичних моделях або статистичних розрахунках, де точність та продуктивність є ключовими.

Крім того, можливість використання різних бібліотек дає можливість вибору відповідно до конкретних потреб проекту. Наприклад, System.Numerics.BigInteger надає зручний та простий інтерфейс для роботи з великими числами, тоді як Math.Gmp.Native або System.Numerics.MPFR можуть бути використані для оптимізації продуктивності у вимірюваних додатках.

Загалом, здатність CSharp підтримувати і вбудовані, і сторонні бібліотеки для роботи з арифметичними операціями робить її потужним і гнучким інструментом для вирішення широкого спектру математичних завдань у сучасному програмуванні.