Basics of Modulus Operator

September 5, 2024

ⓘThis conversation may reflect the link creator’s personalized data, which isn’t shared and can meaningfully change how the model responds.

You said:

In C++, the modulo operator (`%`) is used to find the remainder when one integer is divided by another.

Syntax:

cpp

```cpp
int result = a % b;
```

- **a**: Dividend (the number you're dividing).
- **b**: Divisor (the number you're dividing by).
- **result**: The remainder after division.

Example:

cpp

```cpp
int a = 10;
int b = 3;
int result = a % b;       // result is 1, because 10 divided by 3 leaves a remainder of 1.
```

Important Points:

1. Modulo can only be used with integers.
2. The sign of the result is the same as the sign of the dividend (**`a`**).
3. It's often used to determine divisibility (e.g., if **`a % b == 0`**, then **`a`** is divisible by **`b`**).

This operator is commonly used in algorithms that require circular arrays, checking for even/odd numbers, or handling wrap-around scenarios (like working with indexes in a loop).

You said:

cpp

```
int reverseNumber(int n) {
    int rev = 0;
    while (n > 0) {
        rev = rev * 10 + n % 10;
        n /= 10;
    }
    return rev;
}
```

cpp

```
int countDigits(int n) {
    int count = 0;
    while (n > 0) {
        count++;
        n /= 10;
    }
    return count;
}
```

cpp

```
bool isPalindrome(int n) {
    int original = n, rev = 0;
    while (n > 0) {
        rev = rev * 10 + n % 10;
        n /= 10;
    }
    return original == rev;
}
```

cpp

```cpp
bool isArmstrong(int n) {
    int original = n, sum = 0;
    while (n > 0) {
        int lastdigit = n % 10;
        sum += (lastdigit * lastdigit * lastdigit);
        n /= 10;
    }
    return original == sum;
}
```

cpp

```cpp
void printDivisors(int n) {
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            cout << i << " ";
        }
    }
}
```

Here's a slight correction to your code for printing all divisors optimally:

cpp

```cpp
void printAllDivisorsOpti(int n) {
    for (int i = 1; i <= sqrt(n); i++) {
        if (n % i == 0) {
            cout << i << " "; if (i
            != n / i) {
                cout << n / i << " ";
            }
        }
    }
}
```

This optimized approach runs in `` `O(sqrt(n))` `` , as it checks divisors in pairs.

You said:

ChatGPT said:

cpp

```cpp
void printAllDivisorsOpti(int n) {
    set<int> divisors;
    for (int i = 1; i <= sqrt(n); i++) {
        if (n % i == 0) {
            divisors.insert(i);
            divisors.insert(n / i);
        }
    }
    for (int divisor : divisors) {
        cout << divisor << " ";
    }
}
```

cpp

```cpp
void printPrimes(int n) {
    vector<bool> isPrime(n + 1, true);
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i * i <= n; i++) {
        if (isPrime[i]) {
            for (int j = i * i; j <= n; j += i) {
                isPrime[j] = false;
            }
        }
    }
    for (int i = 2; i <= n; i++) {
        if (isPrime[i]) {
            cout << i << " ";
        }
    }
}
```

You said:

ChatGPT said:

Here’s a corrected version of the function to check if a single number `n` is prime:

cpp

```cpp
void primeNumber(int n) {
    if (n <= 1) {
        cout << "Not prime";
        return;
    }
    int cnt = 0;
```

```
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            if (i * i == n) {
                cnt++;
            } else {
                cnt += 2;
            }
        }
    }
    if (cnt == 2) {
        cout << "It's prime";
    } else {
        cout << "Not prime";
    }
}
```

```cpp
int gcd(int a, int b) {
    int gcd = 1;
    for (int i = 1; i <= min(a, b); i++) {
        if (a % i == 0 && b % i == 0) {
            gcd = i;
        }
    }
    return gcd;
}
```

You said:

ChatGPT said:

```cpp
int gcd(int a, int b) {
    int gcd = 1;
    int i = min(a, b);
    while (i > 0) {
        if (a % i == 0 && b % i == 0) {
            gcd = i;
            break;
        }
        i--;
    }
    return gcd;
}
```

You said:

cpp

```cpp
int gcd(int a, int b) {
    while (b > 0) {
        int temp = a % b;
        a = b;
        b = temp;
    }
    return a;
}
```

Here's the corrected and complete snippet for calculating x^n using exponentiation by squaring:

cpp

```cpp
double power(double x, int n) {
    double ans = 1;
    while (n > 0) {
          if (n % 2 == 1) {
             ans *= x;
        }    x
        * =
        x ;
    }   n
    re/t=urn ans;
}       2 ;
```

cpp

```cpp
void printPrimes(int n) {
    for (int num = 2; num <= n; num++) {
        bool isPrime = true;
        for (int i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                isPrime = false;
                break;
            }
        }
        if (isPrime) {
            cout << num << " ";
        }
```

```
        }
    }
```

```
cpp
```

```cpp
    void printPrimes(int n) {
        vector<bool> isPrime(n + 1, true);
        isPrime[0] = isPrime[1] = false;

        for (int i = 2; i * i <= n; i++) {
            if (isPrime[i]) {
                for (int j = i * i; j <= n; j += i) {
                    isPrime[j] = false;
                }
            }
        }

        for (int i = 2; i <= n; i++) {
            if (isPrime[i]) {
                cout << i << " ";
            }
        }
    }
```

Brute Force: $O(n\sqrt{n})$

Sieve of Eratosthenes: $O(n \log\log n)$

Get started with ChatGPT