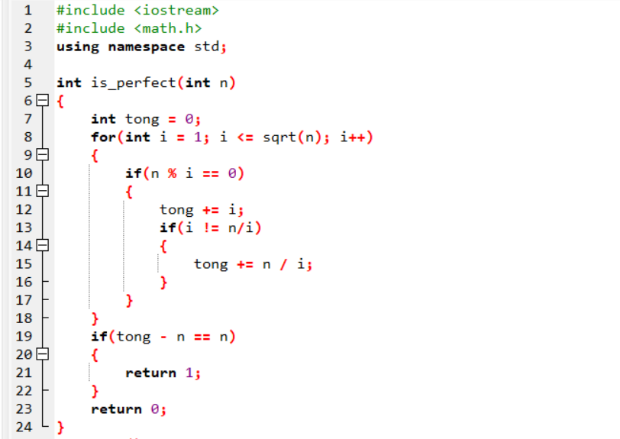
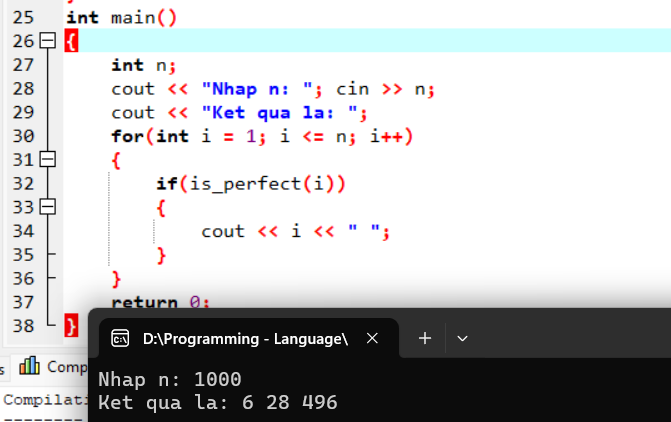
**1. Thuật Toán Cơ Bản**

Số hoàn hảo là số mà tổng các ước thực sự của nó (không tính chính nó) bằng chính nó. Ví dụ N = 28 có các ước 1, 2, 4, 7, 14, 28 thì các ước thực sự là 1, 2, 4, 7, 14 có tổng cũng bằng 28 nên 28 là số hoàn hảo

Để kiểm tra N là số hoàn hảo bạn chỉ cần tính tổng ước của N sau đó loại bỏ ước N và so sánh với N

Code 1:

  
Giải thích:

**- Dòng 5:** Hàm này nhận vào một số nguyên n và trả về kết quả kiểm tra xem n có phải là số hoàn hảo hay không.

**- Dòng 7:** Biến tong dùng để lưu tổng của tất cả các ước của n.

**- Dòng 8:**

+ Vòng lặp này chạy từ 1 đến sqrt(n) (căn bậc hai của n).

+ Vì ước số luôn xuất hiện theo cặp, nếu i là ước của n, thì n/i cũng là ước của n. Ví dụ, nếu n = 28, thì các cặp ước là 1 và 28, 2 và 14, 4 và 7.

**- Dòng 10:** if (n % i == 0): Kiểm tra nếu i là ước của n (tức là n chia hết cho i), thì cộng i vào biến tong.

**- Dòng 13:** if (i != n / i): Điều kiện này đảm bảo rằng chúng ta không cộng hai lần cùng một ước nếu i và n/i bằng nhau (trường hợp này xảy ra khi n là số chính phương). Ví dụ, với n = 36, i = 6, thì n / i cũng là 6, và chỉ cần cộng một lần.

**- Dòng 19 – 23:** Sau khi tính tổng các ước của n, trừ n ra khỏi tổng, vì tổng hiện tại bao gồm cả n.

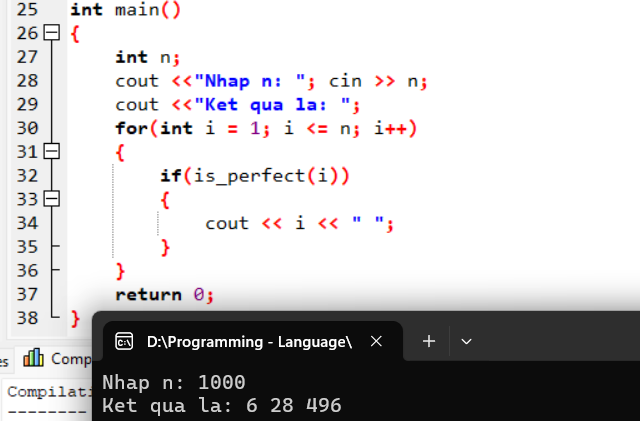
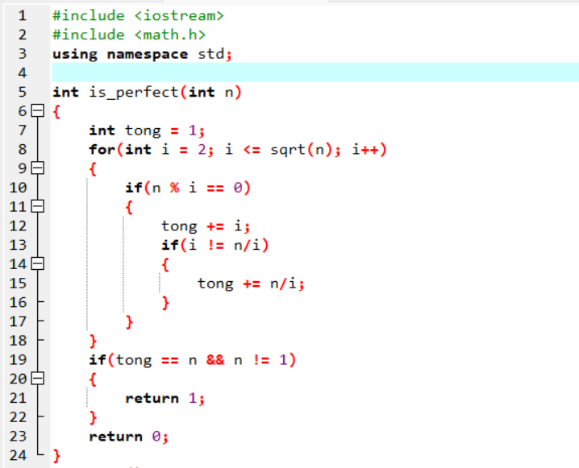
Nếu tổng sau khi trừ bằng đúng n, thì n là số hoàn hảo và hàm trả về 1. Ngược lại, nếu tổng khác n, hàm trả về 0.

Ví dụ: Với n = 28, tổng các ước là 1 + 2 + 4 + 7 + 14 = 28, nên 28 là số hoàn hảo.

=> Code trên tính tổng ước của N sau đó trừ đi chính nó vì ta đang muốn tính các ước mà không xét chính nó.

Bạn cũng có thể bỏ qua ước 1 để không xét ước N như sau, nhưng chú ý khi N = 1

Code 2 :

  
Giải thích:

**- Dòng 7:** Biến tong được khởi tạo là 1, vì 1 là ước số của mọi số (trừ chính số 1 ra), và ta không cần lặp để tìm ước là 1.

**- Dòng 8:** Ta chỉ cần kiểm tra các số từ 2 đến sqrt(n) bởi vì nếu i là ước của n, thì cặp ước của nó là n/i cũng là ước của n.

**- Dòng 19 – 23:**

+ Sau khi tính tổng các ước của n, nếu tổng này đúng bằng n và n khác 1 (vì 1 không được coi là số hoàn hảo), thì trả về 1 (nghĩa là n là số hoàn hảo).

+ Ngược lại, nếu tổng khác n, trả về 0 (không phải số hoàn hảo).

**2. Định Lý Euclid - Euler**

Để kiểm tra số hoàn hảo bạn có thể sử dụng code ở mục 1, tuy nhiên nó chỉ phù hợp khi số N bạn cần kiểm tra ≤ 1012. Trong trường hợp cần kiểm tra số N lên tới 1018 thì code trên chạy rất lâu mới ra kết quả.

Bạn thử viết hàm kiểm tra số hoàn hảo như mục 1 với số long long và thử chạy với  N = 1018 để kiểm tra.

Phương pháp tối ưu hơn để kiểm tra số hoàn hảo đó là sử dụng định lý Euclid - Euler. Định lý này phát biểu như sau

Nếu *p* là số nguyên tố và *2p- 1* cũng là số nguyên tố thì : *2p-1\* (2p - 1)*sẽ là một số hoàn hảo.

Ví dụ :

p = 2 là số nguyên tố, và 22 - 1 cũng là số nguyên tố nên : 22-1 \* (22 - 1) = 6 là số hoàn hảo

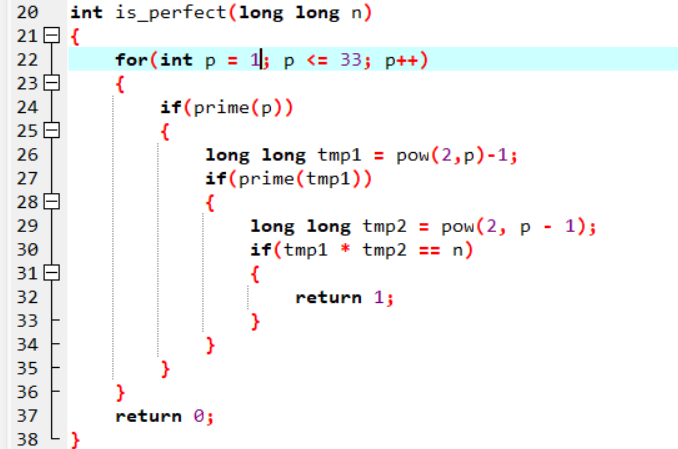
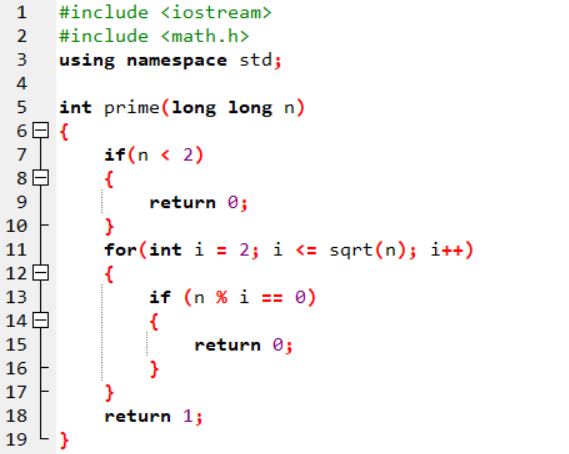
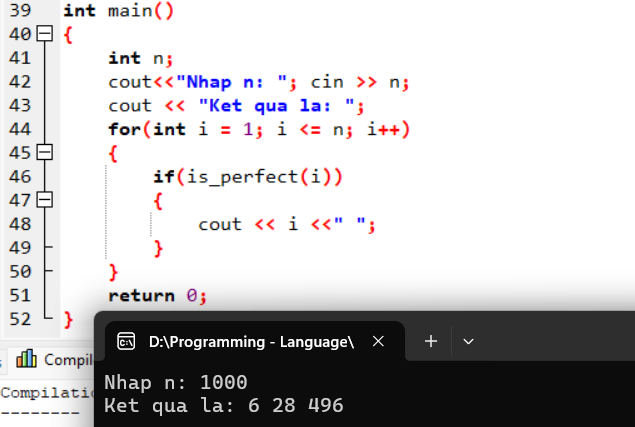
p = 3 là số nguyên tố, và 23 - 1 cũng là số nguyên tố nên : 23-1 \* (23 - 1) = 4 \* 7 = 28 là số hoàn hảo

Vậy để kiểm tra được hết các số hoàn hảo từ 1 tới giới hạn lớn nhất của số long long (9.1018) thì bạn cần duyệt các số nguyên tố p lớn tới đâu ?

Nhận thấy tích *2p-1\* (2p - 1)* ~ 22p sẽ phải lên tới 9.1018 ~ 263 => 2p ~ 63 nên bạn chỉ cần duyệt p tới 32, 33 là đủ rồi.

Code này sẽ sinh ra tất cả các số hoàn hảo từ 1 tới 9.1018 và so sánh với N

Code 1:

Giải thích:

**- Dòng 7:** Nếu n < 2, trả về 0 (không phải số nguyên tố).

**- Dòng 11 – 18**

+ Vòng lặp for từ 2 đến sqrt(n) kiểm tra xem n có chia hết cho bất kỳ số nào trong khoảng đó hay không. Nếu có, n không phải là số nguyên tố và hàm trả về 0.

+ Nếu không có số nào chia hết cho n, thì n là số nguyên tố, hàm trả về 1.

**- Dòng 22:** Chương trình chỉ kiểm tra đến 33 vì với giá trị p > 33, số hoàn hảo Mersenne sẽ quá lớn.

**- Dòng 24 – 37:**  
+ Với mỗi giá trị p, nếu p là số nguyên tố (kiểm tra qua hàm prime(p)), thì tiếp tục xét các bước sau.

+ Tính toán tmp1 = 2^p - 1. Đây là số nguyên tố Mersenne cần kiểm tra.

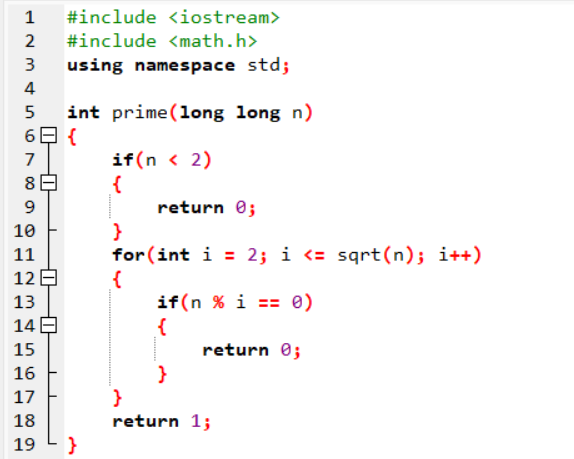
+ Nếu tmp1 là số nguyên tố (kiểm tra bằng hàm prime(tmp1)), tiếp tục bước tiếp theo.

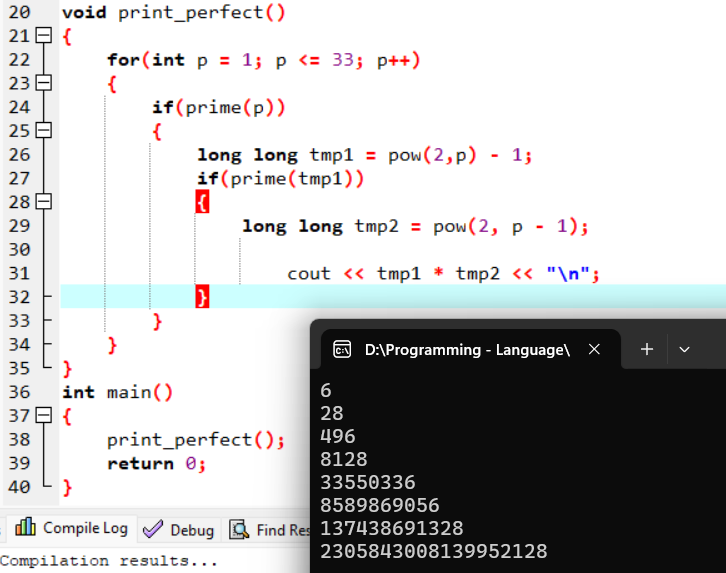
* Tính tmp2 = 2^{p-1}.
* Kiểm tra nếu tích tmp1 \* tmp2 bằng với số cần kiểm tra n, thì n là số hoàn hảo, trả về 1.
* Nếu không, hàm tiếp tục vòng lặp để kiểm tra các giá trị p tiếp theo.

+ Trả về 0 nếu không tìm thấy số hoàn hảo

**Ví dụ với p = 3:**

* p = 3 là số nguyên tố.
* tmp1 = 2^3 - 1 = 7, là số nguyên tố.
* tmp2 = 2^{3-1} = 2^2 = 4.
* tmp1 \* tmp2 = 7 \* 4 = 28, nên 28 là số hoàn hảo.

Code 2: Bạn cũng có thể in ra các số hoàn hảo từ 1 tới 9.1018

  
Giải thích:

- Dòng 7 – 18:

+ Nếu n < 2, trả về 0 (vì các số nhỏ hơn 2 không phải là số nguyên tố).

+ Vòng lặp chạy từ 2 đến căn bậc hai của n, kiểm tra xem n có chia hết cho số nào trong khoảng này hay không. Nếu có, trả về 0 (không phải là số nguyên tố).

+ Nếu không có số nào chia hết cho n, trả về 1 (đây là số nguyên tố).

- Dòng 20 – 31:

* **Mục đích**: Hàm này tìm và in ra các số hoàn hảo Mersenne.
* **Vòng lặp for từ 1 đến 33**:
  + Với mỗi p, nếu p là số nguyên tố (kiểm tra qua hàm prime(p)), tiếp tục các bước sau:
    - Tính tmp1 = 2^p - 1. Đây là số nguyên tố Mersenne tiềm năng.
    - Kiểm tra nếu tmp1 là số nguyên tố (qua hàm prime(tmp1)):
      * Tính tmp2 = 2^{p-1}.
      * Nếu đúng, in ra kết quả: tmp1 \* tmp2 là số hoàn hảo Mersenne.

**Ví dụ với p = 3:**

* p = 3 là số nguyên tố.
* tmp1 = 2^3 - 1 = 7, là số nguyên tố.
* tmp2 = 2^{3-1} = 4.
* Tích tmp1 \* tmp2 = 7 \* 4 = 28, vậy 28 là số hoàn hảo và được in ra.

Nhận xét : từ 1 tới 9.1018cũng chỉ có 8 số hoàn hảo.