**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**CÁC KIẾN THỨC CƠ BẢN TRONG LẬP TRÌNH THI ĐẤU**

MỤC LỤC

[Chương 1 HỆ ĐẾM – NUMERAL SYSTEM 9](#_Toc139281890)

[1.1 Hệ đếm thập phân – Hệ đếm cơ số 10 9](#_Toc139281891)

[1.1.1 Khái niệm hệ đếm thập phân 9](#_Toc139281892)

[1.1.2 Nhắc lại các kiểu số nguyên cơ sở trong 9](#_Toc139281893)

[1.1.3 Các kiểu tựa số nguyên trong 11](#_Toc139281894)

[1.1.4 Ký tự số 11](#_Toc139281895)

[1.1.5 Ký tự alphabet 11](#_Toc139281896)

[1.1.6 Hàm xuất mã ascii của một số ký tự 13](#_Toc139281897)

[1.1.7 Hàm chuyển đổi một ký tự số thành số nguyên 14](#_Toc139281898)

[1.1.8 Hàm chuyển đổi một số nguyên thành ký tự số 16](#_Toc139281899)

[1.2 Hệ đếm nhị phân – Hệ đếm cơ số 2 17](#_Toc139281900)

[1.2.1 Khái niệm hệ đếm nhị phân 17](#_Toc139281901)

[1.2.2 Chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm nhị phân 18](#_Toc139281902)

[1.2.3 Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân 23](#_Toc139281903)

[1.2.4 Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân cải tiến 28](#_Toc139281904)

[1.3 Hệ đếm thập lục phân – Hệ đếm cơ số 16 30](#_Toc139281905)

[1.3.1 Khái niệm hệ đếm thập lục phân 30](#_Toc139281906)

[1.3.2 Chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm thập lục phân 31](#_Toc139281907)

[1.3.3 Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân 33](#_Toc139281908)

[1.3.4 Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân cải tiến 37](#_Toc139281909)

[1.3.5 Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân 39](#_Toc139281910)

[1.3.6 Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập lục phân 43](#_Toc139281911)

[1.4 Hệ đếm tổng quát 46](#_Toc139281912)

[1.4.1 Khái niệm hệ đếm tổng quát 46](#_Toc139281913)

[1.4.2 Hệ đếm cơ số b 46](#_Toc139281914)

[1.4.3 Các hệ đếm phổ biến 47](#_Toc139281915)

[1.4.4 Chuyển đổi giá trị trong biểu diễn cơ số b sang cơ số 10 47](#_Toc139281916)

[1.4.5 Chuyển đổi giá trị trong biểu diễn cơ số 10 sang cơ số b 47](#_Toc139281917)

[1.4.6 Chuyển đổi giá trị trong biểu diễn cơ số b sang cơ số c 47](#_Toc139281918)

[1.5 Bài tập hệ đếm 48](#_Toc139281919)

[Chương 2 SỐ NGUYÊN TỐ – PRIME NUMBER 54](#_Toc139281920)

[2.1 Khái niệm số nguyên tố 54](#_Toc139281921)

[2.2 Khái niệm hợp số 54](#_Toc139281922)

[2.3 Kiếm tra tính nguyên tố chính xác 54](#_Toc139281923)

[2.3.1 Giải thuật kiểm tra nguyên tố theo định nghĩa 55](#_Toc139281924)

[2.3.2 Giải thuật kiểm tra số nguyên tố cải tiến 56](#_Toc139281925)

[2.3.3 Cải tiến dựa vào định lý toán học 57](#_Toc139281926)

[2.3.4 Cải tiến dựa vào nhận xét logic kiểu của dân tin học 59](#_Toc139281927)

[2.4 Kiếm tra tính nguyên tố theo xác suất 62](#_Toc139281928)

[2.4.1 Phương pháp Fermat 62](#_Toc139281929)

[2.4.2 Phương pháp Miller – Rabin 63](#_Toc139281930)

[2.4.3 Phương pháp Solovay – Strassen 63](#_Toc139281931)

[2.4.3.1 Ký hiệu Legendre 63](#_Toc139281932)

[2.4.3.2 Tiêu chuẩn Euler 64](#_Toc139281933)

[2.4.3.3 Ký hiệu Jacobi 64](#_Toc139281934)

[2.5 Tìm các số nguyên tố trong đoạn 64](#_Toc139281935)

[2.6 Thuật toán sàng Eratosthene – Sieve of Eratosthenes 66](#_Toc139281936)

[2.6.1 Minh họa từng bước thuật toán Eratosthenes 67](#_Toc139281937)

[2.6.2 Thuật toán sàng Eratosthenes 70](#_Toc139281938)

[2.6.3 Hàm cài đặt thuật toán sàng Eratosthenes 70](#_Toc139281939)

[2.7 Phân tích thừa số nguyên tố 73](#_Toc139281940)

[2.7.1 Giải thuật phân tích thừa số nguyên tố ngây thơ 74](#_Toc139281941)

[2.7.2 Cải tiến giải thuật phân tích thừa số nguyên tố ngây thơ 77](#_Toc139281942)

[2.7.3 Phân tích thừa số nguyên tố với sàng Eratosthenes 81](#_Toc139281943)

[2.8 Sàng Eratosthenes trên đoạn 88](#_Toc139281944)

[2.9 Định lý Wilson 89](#_Toc139281945)

[2.10 Bài tập số nguyên tố 89](#_Toc139281946)

[Chương 3 ƯỚC SỐ - BỘI SỐ 91](#_Toc139281947)

[3.1 Lý thuyết chia hết 91](#_Toc139281948)

[3.2 Lý thuyết đồng dư 92](#_Toc139281949)

[3.2.1 Định nghĩa 92](#_Toc139281950)

[3.2.2 Một số tính chất đồng dư 92](#_Toc139281951)

[3.3 Số lượng ước số của một số nguyên dương 93](#_Toc139281952)

[3.3.1 Giải thuật ngây thơ đếm số lượng ước số 94](#_Toc139281953)

[3.3.2 Giải thuật đếm số lượng ước số dựa vào thuật toán phân tích thừa số nguyên tố 94](#_Toc139281954)

[3.3.3 Giải thuật đếm số lượng ước số dựa vào thuật toán phân tích thừa số nguyên tố trong đoạn 98](#_Toc139281955)

[3.3.4 Giải thuật dựa vào phân tích thừa số nguyên tố với sàng Eratosthenes 99](#_Toc139281956)

[3.4 Tổng các ước số của một số nguyên dương 100](#_Toc139281957)

[3.4.1 Giải thuật thơ ngây tính tổng ước số 100](#_Toc139281958)

[3.4.2 Giải thuật tính tổng ước số dựa vào thuật toán phân tích thừa số nguyên tố 101](#_Toc139281959)

[3.4.3 Giải thuật tính tổng ước số dựa vào thuật toán phân tích thừa số nguyên tố trong đoạn 104](#_Toc139281960)

[3.4.4 Giải thuật tính tổng ước số dựa vào phân tích thừa số nguyên tố với sàng Eratosthenes 105](#_Toc139281961)

[3.5 Ước số chung lớn nhất của hai số nguyên dương 106](#_Toc139281962)

[3.5.1 Phương pháp hiệu 106](#_Toc139281963)

[3.5.2 Phương pháp chia lấy dư 107](#_Toc139281964)

[3.5.3 Phương pháp Euclid 108](#_Toc139281965)

[3.6 Bội số chung nhỏ nhất của hai số nguyên dương 111](#_Toc139281966)

[3.7 Hàm phi Euler 112](#_Toc139281967)

[3.8 Bài tập ước số, bội số 113](#_Toc139281968)

[Chương 4 LÝ THUYẾT TẬP HỢP 114](#_Toc139281969)

[4.1 Các phép toán trên tập hợp 114](#_Toc139281970)

[4.2 Các tính chất của phép toán trên tập hợp 114](#_Toc139281971)

[4.3 Tích Đề-các của các tập hợp 115](#_Toc139281972)

[4.4 Nguyên lí cộng 115](#_Toc139281973)

[4.5 Nguyên bù trừ 116](#_Toc139281974)

[4.6 Nguyên lí nhân 116](#_Toc139281975)

[4.7 Chỉnh hợp lặp 116](#_Toc139281976)

[4.8 Chỉnh hợp không lặp 117](#_Toc139281977)

[4.9 Hoán vị 117](#_Toc139281978)

[4.10 Tổ hợp 117](#_Toc139281979)

[4.11 Bài tập lý thuyết tập hợp 119](#_Toc139281980)

[Chương 5 CÁC SỐ ĐẶC BIỆT 120](#_Toc139281981)

[5.1 Số fibonacci 120](#_Toc139281982)

[5.1.1 Phương pháp sử dụng bộ nhớ phụ tính số hạng thứ n của dãy fibonacci 120](#_Toc139281983)

[5.1.2 Phương pháp không sử dụng bộ nhớ phụ tính số hạng thứ n của dãy fibonacci 121](#_Toc139281984)

[5.1.3 Phương pháp đệ quy tính số hạng thứ n của dãy fibonacci 122](#_Toc139281985)

[5.1.4 Các tính chất của số fibonacci 123](#_Toc139281986)

[5.2 Số Catalan 123](#_Toc139281987)

[5.2.1 Tính số hạng thứ n của dãy Catalan theo định nghĩa 124](#_Toc139281988)

[5.2.2 Tính số hạng thứ n của dãy Catalan bằng phương pháp sử dụng bộ nhớ phụ 125](#_Toc139281989)

[5.2.3 Tính số hạng thứ n của dãy Catalan bằng phương pháp không sử dụng bộ nhớ phụ 126](#_Toc139281990)

[5.2.4 Tính số hạng thứ n của dãy Catalan bằng phương pháp đệ quy 128](#_Toc139281991)

[5.3 Số chính phương 128](#_Toc139281992)

[5.4 Số phong phú 128](#_Toc139281993)

[5.5 Bài tập các số đặc biệt 128](#_Toc139281994)

[Chương 6 MỘT SỐ THUẬT TOÁN 129](#_Toc139281995)

[6.1 Phép nhân Ai cập 129](#_Toc139281996)

[6.1.1 Giải thuật nhân Ai cập 129](#_Toc139281997)

[6.1.2 Hàm cài đặt xử lý trên số nguyên 130](#_Toc139281998)

[6.1.3 Hàm cài đặt xử lý trên bit 132](#_Toc139281999)

[6.1.4 Hàm cài đặt đệ quy 133](#_Toc139282000)

[6.2 Tính với 133](#_Toc139282001)

[6.2.1 Hàm cài đặt xử lý trên số nguyên 134](#_Toc139282002)

[6.2.2 Hàm cài đặt đệ quy 134](#_Toc139282003)

[6.3 Tính lũy thừa , tính mũ 135](#_Toc139282004)

[6.3.1 Giải thuật tự nhiên tính lũy thừa, tính mũ 136](#_Toc139282005)

[6.3.2 Phương pháp đệ quy tính lũy thừa, tính mũ 137](#_Toc139282006)

[6.3.3 Phương pháp tính lũy thừa nhanh, tính mũ nhanh 138](#_Toc139282007)

[6.3.4 Phương pháp tính lũy thừa nhanh, , tính mũ nhanh bằng đệ quy 140](#_Toc139282008)

[6.4 Tính 141](#_Toc139282009)

[6.4.1 Giải thuật tự nhiên 142](#_Toc139282010)

[6.4.2 Giải thuật cải tiến 143](#_Toc139282011)

[6.4.3 Giải thuật cải tiến đệ quy 144](#_Toc139282012)

[6.5 Một số đẳng thức cần biết 144](#_Toc139282013)

[6.5.1 Tổng từ 1 tới n 144](#_Toc139282014)

[6.5.1.1 Phương pháp truyền thống 145](#_Toc139282015)

[6.5.1.2 Sử dụng công thức toán học 145](#_Toc139282016)

[6.5.2 Tổng các bình phương 146](#_Toc139282017)

[6.5.2.1 Phương pháp truyền thống 146](#_Toc139282018)

[6.5.2.2 Sử dụng công thức toán học 147](#_Toc139282019)

[6.5.3 Tổng các lập phương 147](#_Toc139282020)

[6.5.3.1 Phương pháp truyền thống 147](#_Toc139282021)

[6.5.3.2 Sử dụng công thức toán học 148](#_Toc139282022)

[6.5.4 Tổng các số lẻ 149](#_Toc139282023)

[6.5.4.1 Phương pháp truyền thống 149](#_Toc139282024)

[6.5.4.2 Sử dụng công thức toán học 149](#_Toc139282025)

[6.5.5 Tổng các số chẵn 150](#_Toc139282026)

[6.5.5.1 Phương pháp truyền thống 150](#_Toc139282027)

[6.5.5.2 Sử dụng công thức toán học 151](#_Toc139282028)

[6.5.6 Tổng dãy số có giá trị bắt đầu là 2 bước nhảy là 3 151](#_Toc139282029)

[6.5.6.1 Phương pháp truyền thống 152](#_Toc139282030)

[6.5.6.2 Sử dụng công thức toán học 152](#_Toc139282031)

[6.6 Một số bất đẳng thức cần biết 153](#_Toc139282032)

[6.6.1 Bất đẳng thức AM – GM 153](#_Toc139282033)

[6.6.2 Bất đẳng thức Cauchy–Bunyakovsky–Schwarz 153](#_Toc139282034)

[6.7 Bài tập các thuật giải cơ sở 153](#_Toc139282035)

[Chương 7 SỐ NGUYÊN LỚN 155](#_Toc139282036)

[7.1 Biểu diễn số nguyên lớn 155](#_Toc139282037)

[7.2 Các hàm cơ bản xử lý số lớn 155](#_Toc139282038)

[7.2.1 Hàm đổi số thành ký tự 156](#_Toc139282039)

[7.2.2 Hàm đổi ký tự thành số 156](#_Toc139282040)

[7.3 Phép so sánh hai số nguyên lớn 157](#_Toc139282041)

[7.4 Phép cộng hai số nguyên lớn 160](#_Toc139282042)

[7.5 Phép trừ hai số nguyên lớn 163](#_Toc139282043)

[7.6 Phép nhân một số nguyên lớn với một số nguyên nhỏ 166](#_Toc139282044)

[7.7 Phép nhân hai số nguyên lớn 169](#_Toc139282045)

[7.8 Phép chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ 172](#_Toc139282046)

[7.9 Phép chia lấy dư (mod) một số nguyên lớn với một số nguyên nhỏ 173](#_Toc139282047)

[7.10 Phép chia lấy thương nguyên (div) hai số nguyên lớn 175](#_Toc139282048)

[7.11 Phép chia lấy dư (mod) hai số nguyên lớn 177](#_Toc139282049)

[7.12 Tính số Fibonacci thứ 179](#_Toc139282050)

[7.13 Tính giai thừa thứ n 180](#_Toc139282051)

[7.14 Tính số hạng Catalan thứ 182](#_Toc139282052)

[7.15 Bài tập 183](#_Toc139282053)

# HỆ ĐẾM – NUMERAL SYSTEM

## Hệ đếm thập phân – Hệ đếm cơ số 10

### Khái niệm hệ đếm thập phân

Khái niệm: hệ thập phân (hệ cơ số ) dùng kí hiệu .

* Ví dụ 01:
* Ví dụ 02:

### Nhắc lại các kiểu số nguyên cơ sở trong

* Trong có kiểu số nguyên cơ sở.
* int
* unsigned int
* short
* unsigned short
* long
* unsigned long
* long long
* unsigned long long
* Bảng chi tiết các kiểu cơ sở trong .

| STT | Kiểu dữ liệu | Kích thước | Phạm vi biểu diễn |
| --- | --- | --- | --- |
| 1 | , | byte  byte | tới hoặc  tới |
| 2 | , | byte  byte | tới hoặc  tới |
| 3 | ,  , | byte | tới |
| 4 | , | byte | tới |
| 5 | ,  , | byte | tới |
| 6 | , | byte | tới |
| 7 | , | byte | tới |
| 8 |  | byte | tới |

1. Viết chương trình in ra giá trị , của kiểu dữ liệu số nguyên int và signed trong ngôn ngữ .

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. cout << "\n int min: ";
6. cout << numeric\_limits<int>::min();
7. cout << "\n int max: ";
8. cout << numeric\_limits<int>::max();
9. cout << "\n signed min: ";
10. cout << numeric\_limits<signed>::min();
11. cout << "\n signed max: ";
12. cout << numeric\_limits<signed>::max();
13. cout << "\n\n\nKet thuc!!!!!";
14. return 1;
15. }

* Viết với ngôn ngữ .

### Các kiểu tựa số nguyên trong

| STT | Kiểu dữ liệu | Kích thước | Phạm vi biểu diễn |
| --- | --- | --- | --- |
| 1 | , | byte | tới hoặc  tới |
| 2 | , | byte | tới |
| 3 |  | byte | tới |
| 4 |  | byte | tới |

### Ký tự số

Các ký tự số và mã ascii tương ứng

| Ký tự số | Mã ascii |
| --- | --- |
| '0' | 48 |
| '1' | 49 |
| '2' | 50 |
| '3' | 51 |
| '4' | 52 |
| '5' | 53 |
| '6' | 54 |
| '7' | 55 |
| '8' | 56 |
| '9' | 57 |

### Ký tự alphabet

Bảng mã ascii của các ký tự chữ cái hoa:

| Thứ tự | Ký tự | Mã ascii |
| --- | --- | --- |
| 1 | 'A' | 65 |
| 2 | 'B' | 66 |
| 3 | 'C' | 67 |
| 4 | 'D' | 68 |
| 5 | 'E' | 69 |
| 6 | 'F' | 70 |
| 7 | 'G' | 71 |
| 8 | 'H' | 72 |
| 9 | 'I' | 73 |
| 10 | 'J' | 74 |
| 11 | 'K' | 75 |
| 12 | 'L' | 76 |
| 13 | 'M' | 77 |
| 14 | 'N' | 78 |
| 15 | 'O' | 79 |
| 16 | 'P' | 80 |
| 17 | 'U' | 81 |
| 18 | 'R' | 82 |
| 19 | 'S' | 83 |
| 20 | 'T' | 84 |
| 21 | 'U' | 85 |
| 22 | 'V' | 86 |
| 23 | 'W' | 87 |
| 24 | 'X' | 88 |
| 25 | 'Y' | 89 |
| 26 | 'Z' | 90 |

Bảng mã ascii của các ký tự chữ cái thường:

| Thứ tự | Ký tự | Mã ascii |
| --- | --- | --- |
| 1 | 'a' | 97 |
| 2 | 'b' | 98 |
| 3 | 'c' | 99 |
| 4 | 'd' | 100 |
| 5 | 'e' | 101 |
| 6 | 'f' | 102 |
| 7 | 'g' | 103 |
| 8 | 'h' | 104 |
| 9 | 'i' | 105 |
| 10 | 'j' | 106 |
| 11 | 'k' | 107 |
| 12 | 'l' | 108 |
| 13 | 'm' | 109 |
| 14 | 'n' | 110 |
| 15 | 'o' | 111 |
| 16 | 'p' | 112 |
| 17 | 'u' | 113 |
| 18 | 'r' | 114 |
| 19 | 's' | 115 |
| 20 | 't' | 116 |
| 21 | 'u' | 117 |
| 22 | 'v' | 118 |
| 23 | 'w' | 119 |
| 24 | 'x' | 120 |
| 25 | 'y' | 121 |
| 26 | 'z' | 122 |

### Hàm xuất mã ascii của một số ký tự

Định nghĩa hàm xuất mã ascii của các ký tự: '0'.. '9', 'A'.. 'Z', 'a'.. 'z'.

* Viết với ngôn ngữ .

1. void XuatAscii()
2. {
3. for (char c = '0'; c <= '9'; c++)
4. cout << "\n" << c << " " << int(c);
5. for (char c = 'A'; c <= 'Z'; c++)
6. cout << "\n" << c << " " << int(c);
7. for (char c = 'a'; c <= 'z'; c++)
8. cout << "\n" << c << " " << int(c);
9. }

* Viết với ngôn ngữ .

1. Viết chương trình xuất mã ascii của các ký tự : '0'.. '9', 'A'.. 'Z', 'a'.. 'z'.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. void XuatAscii();
4. int main()
5. {
6. XuatAscii();
7. }
8. ...

* Viết với ngôn ngữ .

### Hàm chuyển đổi một ký tự số thành số nguyên

Bảng chuyển đổi ký tự số thành số nguyên.

| Ký tự số | Số nguyên |
| --- | --- |
| '0' | 0 |
| '1' | 1 |
| '2' | 2 |
| '3' | 3 |
| '4' | 4 |
| '5' | 5 |
| '6' | 6 |
| '7' | 7 |
| '8' | 8 |
| '9' | 9 |
| 'A' or 'a' | 10 |
| 'B' or 'b' | 11 |
| 'C' or 'c' | 12 |
| 'D' or 'd' | 13 |
| 'E' or 'e' | 14 |
| 'F' or 'f' | 15 |

Khai báo hàm chuyển đổi một ký tự số thành số nguyên.

1. int toNumber(char);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm toNumber ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm toNumber trả về một số nguyên trong đoạn .

Định nghĩa hàm chuyển đổi một ký tự số thành số nguyên.

1. int toNumber(char c)
2. {
3. if (c == '0')
4. return 0;
5. if (c == '1')
6. return 1;
7. if (c == '2')
8. return 2;
9. if (c == '3')
10. return 3;
11. if (c == '4')
12. return 4;
13. if (c == '5')
14. return 5;
15. if (c == '6')
16. return 6;
17. if (c == '7')
18. return 7;
19. if (c == '8')
20. return 8;
21. if (c == '9')
22. return 9;
23. if (c == 'A' || c == 'a')
24. return 10;
25. if (c == 'B' || c == 'b')
26. return 11;
27. if (c == 'C' || c == 'c')
28. return 12;
29. if (c == 'D' || c == 'd')
30. return 13;
31. if (c == 'E' || c == 'e')
32. return 14;
33. return 15;
34. }

* Viết với ngôn ngữ .

### Hàm chuyển đổi một số nguyên thành ký tự số

Bảng chuyển đổi ký tự số thành số nguyên.

| Ký tự số | Số nguyên |
| --- | --- |
| '0' | 0 |
| '1' | 1 |
| '2' | 2 |
| '3' | 3 |
| '4' | 4 |
| '5' | 5 |
| '6' | 6 |
| '7' | 7 |
| '8' | 8 |
| '9' | 9 |
| 'A' or 'a' | 10 |
| 'B' or 'b' | 11 |
| 'C' or 'c' | 12 |
| 'D' or 'd' | 13 |
| 'E' or 'e' | 14 |
| 'F' or 'f' | 15 |

Khai báo hàm chuyển đổi một số nguyên thành ký tự số.

1. char toChar(int);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm toChar ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm toChar trả về một ký tự.

Định nghĩa hàm chuyển đổi một số nguyên thành ký tự số.

1. char toChar(int n)
2. {
3. char Letter[] = { '0','1','2','3','4','5',
4. '6','7','8','9','A','B','C','D','E','F' };
5. return Letter[n];
6. }

* Viết với ngôn ngữ .

## Hệ đếm nhị phân – Hệ đếm cơ số 2

### Khái niệm hệ đếm nhị phân

Khái niệm: hệ nhị phân (hệ cơ số 2) chỉ dùng hai kí hiệu .

* Ví dụ 01:
* Ví dụ 02:

### Chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm nhị phân

Ý tưởng thuật giải chuyển đổi một giá trị nguyên từ hệ đếm thập phân sang hệ đếm nhị phân.

* Một số nguyên chia cho hoài ta được số .
* Mỗi lần chia cho ta có phần dư.
* Đổi phần dư thành ký tự.
* Ký tự được cộng vào bên trái của chuỗi nhị phân kết quả.

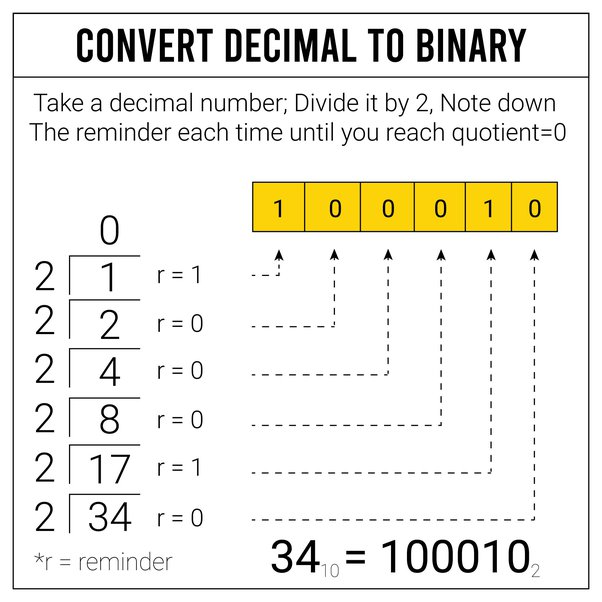
Chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm nhị phân.

* Giá trị trong hệ đếm thập phân: .
* Thực hiện chuyển đổi:

| Giá trị | Dư | Ký tự | Chuỗi nhị phân kết quả |
| --- | --- | --- | --- |
| 1.037 |  |  | "" |
| 518 | 1 | '1' | "1" |
| 259 | 0 | '0' | "01" |
| 129 | 1 | '1' | "101" |
| 64 | 1 | '1' | "1101" |
| 32 | 0 | '0' | "01101" |
| 16 | 0 | '0' | "001101" |
| 8 | 0 | '0' | "0001101" |
| 4 | 0 | '0' | "00001101" |
| 2 | 0 | '0' | "000001101" |
| 1 | 0 | '0' | "0000001101" |
| 0 | 1 | '1' | "10000001101" |

* Giá trị trong hệ đếm thập phân:

Hình vẽ minh họa chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm nhị phân.



Khai báo hàm chuyển đổi một giá trị trong cơ số 10 thành một chuỗi ký tự nhị phân:

1. string DecimalToBinary(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm DecimalToBinary ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm DecimalToBinary trả về một chuỗi ký tự nhị phân.

Định nghĩa hàm chuyển đổi một giá trị trong cơ số 10 thành một chuỗi ký tự nhị phân.

1. string DecimalToBinary\_1(long long n)
2. {
3. string temp;
4. while (n > 0)
5. {
6. if (n % 2 != 0)
7. temp = '1' + temp;
8. else
9. temp = '0' + temp;
10. n = n / 2;
11. }
12. return temp;
13. }

* Viết với ngôn ngữ .

Hàm cài đặt có độ phức tạp: do trong mỗi bước lặp giải thuật chia cho .

* Xem xét câu lệnh:

1. temp = '1' + temp;

* Biến temp có kiểu chuỗi ký tự (string).
* '1' được hiểu là ký tự biểu diễn chữ số một.
* Câu lệnh trên là câu lệnh thực hiện hiện phép cộng ký tự '1' vào bên trái chuỗi ký tự temp.

Chạy từng bước với .

* Điều kiện lặp Đ
  + If S
  + Else
* Điều kiện lặp Đ
  + If S
  + Else
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp S
* Xuất

Hàm trên có thể viết gọn gàng lại như sau bằng cách sử dụng mảng các ký tự nhị phân '0','1' (mảng ).

* Phần từ đầu tiên có chỉ số có giá trị là ký tự '0'.
* Phần từ thứ hai có chỉ số có giá trị là ký tự '1'.

1. string DecimalToBinary\_2(long long n)
2. {
3. string temp;
4. while (n > 0)
5. {
6. temp = toChar(n % 2) + temp;
7. n /= 2;
8. }
9. return temp;
10. }

* Viết với ngôn ngữ .

Chạy từng bước với .

* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp S
* Xuất

Hàm trên có thể viết lại bằng cách xử lý trên bit.

1. string DecimalToBinary\_3(long long n)
2. {
3. string temp;
4. while (n)
5. {
6. if (n & 1)
7. temp = '1' + temp;
8. else
9. temp = '0' + temp;
10. n >>= 1;
11. }
12. return temp;
13. }

* Viết với ngôn ngữ .

1. Viết chương trình nhập một số nguyên dương và đổi số nguyên dương sang hệ đếm nhị phân.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. char toChar(int);
4. string DecimalToBinary(long long);
5. int main()
6. {
7. long long n;
8. cout << "Nhap n (n <= 10^9): ";
9. cin >> n;
10. cout << "\nBieu dien so thap phan ";
11. cout << n << " o he co so 2 la: ";
12. cout << DecimalToBinary(n);
13. cout << "\n\n\nKet thuc!!!!";
14. return 1;
15. }
16. ...

* Lưu ý: hàm toChar và DecimalToBinary chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

### Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân

Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm nhị phân:
* Thực hiện chuyển đổi:
* Giá trị trong hệ đếm thập phân:

Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm nhị phân:
* Thực hiện chuyển đổi:
* Giá trị trong hệ đếm thập phân:

Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm nhị phân:
* Thực hiện chuyển đổi:
* Giá trị trong hệ đếm thập phân:

Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm nhị phân:
* Đánh chỉ mục trong chuỗi nhị phân:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

* Chữ số hàng cao nhất trong chuỗi nhị phân nằm ở đầu chuỗi.
* Đảo chuỗi nhị phân:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

* Lũy thừa :

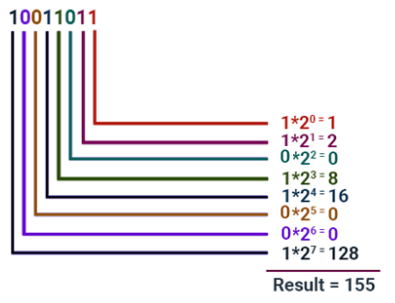
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |

* Chuyển đổi nhị phân sang thập phân:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |

* Giá trị trong hệ đếm thập phân:

Hình vẽ minh họa chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân



Khai báo hàm chuyển đổi một chuỗi ký tự nhị phân thành một giá trị trong hệ cơ số .

1. long long BinaryToDecimal(string);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm BinaryToDecimal ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm BinaryToDecimal trả về một số nguyên kiểu long long.

Định nghĩa hàm chuyển đổi một chuỗi ký tự nhị phân thành một giá trị trong hệ cơ số .

1. long long BinaryToDecimal(string n)
2. {
3. reverse(n.begin(), n.end());
4. long long temp = 0;
5. long long twopower = 1;
6. for (int i = 0; i < n.length(); i++)
7. {
8. temp += twopower \* toNumber(n[i]);
9. twopower \*= 2;
10. }
11. return temp;
12. }

* Viết với ngôn ngữ .

Chạy từng bước với .

* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp S
* Xuất
* Giá trị trong hệ đếm thập phân:

1. Viết chương trình nhập một chuỗi nhị phân và chuyển đổi chuỗi nhị phân ra giá trị trong hệ thập phân.

* Viết với ngôn ngữ .

1. #include <iostream>
2. #include <string>
3. using namespace std;
4. int toNumber(char);
5. long long BinaryToDecimal(string);
6. int main()
7. {
8. string sBinary;
9. cout << "Nhap chuoi nhi phan: ";
10. cin >> sBinary;
11. cout << "\nGia tri thap phan cua chuoi ";
12. cout << sBinary << " la: ";
13. cout << BinaryToDecimal(sBinary);
14. cout << "\n\n\nKet thuc!!!!";
15. return 1;
16. }
17. ...

* Lưu ý: hàm toNumber và BinaryToDecimal chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

### Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân cải tiến

Một cách chuyển đổi khác:

* Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập phân.
* Chỉ mục trong chuỗi nhị phân:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

* Chữ số hàng cao nhất trong chuỗi nhị phân nằm ở đầu chuỗi.
* Chuyển đổi nhị phân sang thập phân:

| Chỉ số | Ký tự | Số nguyên | Giá trị thập phân | Giải thích |
| --- | --- | --- | --- | --- |
|  |  |  | 0 | 0 |
| 0 | '1' | 1 | 1 |  |
| 1 | '0' | 0 | 2 |  |
| 2 | '0' | 0 | 4 |  |
| 3 | '0' | 0 | 8 |  |
| 4 | '0' | 0 | 16 |  |
| 5 | '0' | 0 | 32 |  |
| 6 | '0' | 0 | 64 |  |
| 7 | '1' | 1 | 129 |  |
| 8 | '1' | 1 | 259 |  |
| 9 | '0' | 0 | 518 |  |
| 10 | '1' | 1 | 1.037 |  |

Định nghĩa hàm chuyển đổi một chuỗi ký tự nhị phân thành một giá trị trong hệ cơ số với điều kiện ràng buộc không đảo chuỗi nhị phân ban đầu.

1. long long BinaryToDecimal(string n)
2. {
3. long long temp = 0;
4. for (int i = 0; i < n.length(); i++)
5. {
6. temp = temp \* 2;
7. temp = temp + toNumber(n[i]);
8. }
9. return temp;
10. }

* Viết với ngôn ngữ .

Chạy từng bước với .

* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp S
* Xuất
* Giá trị trong hệ đếm thập phân:

1. Viết chương trình nhập một số nguyên dương và đổi số nguyên dương sang hệ đếm nhị phân với điều kiện ràng buộc không đảo chuỗi nhị phân ban đầu.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. char toChar(int);
4. string DecimalToBinary(long long);
5. int main()
6. {
7. long long n;
8. cout << "Nhap n (n <= 10^9): ";
9. cin >> n;
10. cout << "\nBieu dien so thap phan ";
11. cout << n << " o he co so 2 la: ";
12. cout << DecimalToBinary(n);
13. cout << "\n\n\nKet thuc!!!!";
14. return 1;
15. }
16. ...

* Lưu ý: hàm toChar và DecimalToBinary chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

## Hệ đếm thập lục phân – Hệ đếm cơ số 16

### Khái niệm hệ đếm thập lục phân

Khái niệm: hệ đếm cơ số (hệ thập lục phân , hệ cơ số 16, hệ hexa) dùng kí hiệu . Trong đó có các giá trị tương ứng là .

* Ví dụ 01:
* Ví dụ 02:

### Chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm thập lục phân

Chuyển đổi giá trị từ hệ đếm thập phân sang hệ đếm thập lục phân.

* Giá trị trong hệ đếm thập phân:
* Thực hiện chuyển đổi:

|  |  |  |  |
| --- | --- | --- | --- |
| Giá trị | Dư | Ký tự | Chuỗi thập lục phân kết quả |
| 15.648 |  |  | "" |
| 978 | 0 | '0' | "0" |
| 61 | 2 | '2' | "20" |
| 3 | 13 | 'D' | "D20" |
| 0 | 3 | '3' | "3D20" |

* Giá trị trong hệ đếm thập phân:

Khai báo hàm chuyển đổi giá trị hệ đếm thập phân sang hệ đếm thập lục phân.

1. string DecimalToHexa(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm DecimalToHexa ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm DecimalToHexa trả về một chuỗi ký tự.

Định nghĩa hàm chuyển đổi giá trị hệ đếm thập phân sang hệ đếm thập lục phân.

1. string DecimalToHexa(long long n)
2. {
3. string temp;
4. while (n > 0)
5. {
6. temp = toChar(n % 16) + temp;
7. n = n / 16;
8. }
9. return temp;
10. }

* Viết với ngôn ngữ .

Chạy từng bước với .

* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp S
* Xuất

1. Viết chương trình nhập một số nguyên dương và đổi số nguyên dương sang hệ đếm thập lục phân.

* Viết với ngôn ngữ .

1. #include <iostream>
2. #include <string>
3. using namespace std;
4. char toChar(int);
5. string DecimalToHexa(long long);
6. int main()
7. {
8. long long n;
9. cout << "Nhap n (n <= 10^9): ";
10. cin >> n;
11. cout << "\nBieu dien so thap phan ";
12. cout << n << " o he co so 16 la: ";
13. cout << DecimalToHexa(n);
14. cout << "\n\n\nKet thuc!!!!";
15. return 1;
16. }
17. ...

* Lưu ý: hàm toChar và DecimalToHexa chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

### Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân

Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm nhị phân:
* Thực hiện chuyển đổi:
* Giá trị trong hệ đếm thập phân:

Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm nhị phân:
* Thực hiện chuyển đổi:
* Giá trị trong hệ đếm thập phân:

Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân.

* Giá trị trong hệ đếm thập lục phân:
* Thực hiện chuyển đổi:
  + Chuỗi thập lục phân:
  + Đánh chỉ mục trong chuỗi thập lục phân:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| A | 7 | 0 | D |

* + Chữ số cao nhất trong chuỗi thập lục phân nằm ở đầu chuỗi.
  + Đảo chuỗi thập lục phân:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| D | 0 | 7 | A |

* + Lũy thừa :

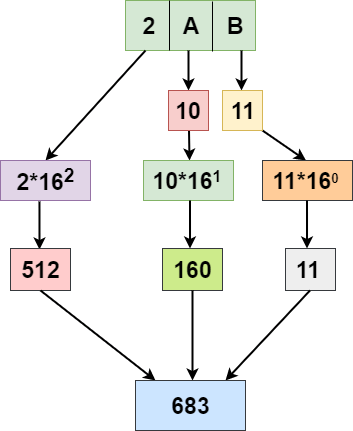
|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| D | 0 | 7 | A |
|  |  |  |  |

* + Chuyển đổi thập lục phân sang thập phân:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| D | 0 | 7 | A |
|  |  |  |  |

* Giá trị trong hệ đếm thập phân:

Hình vẽ minh họa



Khai báo hàm chuyển đổi một chuỗi ký tự thập lục phân thành một giá trị trong hệ cơ số .

1. long long HexaToDecimal(string);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm DecimalToHexa ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm HexaToDecimal trả về một số nguyên có kiểu long long.

Định nghĩa hàm chuyển đổi một chuỗi ký tự thập lục phân thành một giá trị trong hệ cơ số .

1. long long HexaToDecimal(string n)
2. {
3. reverse(n.begin(), n.end());
4. long long temp = 0;
5. long long sixteenpower = 1;
6. for (int i = 0; i < n.length(); i++)
7. {
8. temp += sixteenpower \* toNumber(n[i]);
9. sixteenpower \*= 16;
10. }
11. return temp;
12. }

* Viết với ngôn ngữ .

1. Viết chương trình nhập một chuỗi thập lục phân và chuyển đổi chuỗi thập lục phân ra giá trị trong hệ thập phân.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. int toNumber(char);
4. long long HexaToDecimal(string);
5. int main()
6. {
7. string sHexa;
8. cout << "Nhap chuoi thap luc phan: ";
9. cin >> sHexa;
10. cout << "\nGia tri thap phan cua chuoi ";
11. cout << sHexa << " la: ";
12. cout << HexaToDecimal(sHexa);
13. cout << "\n\n\nKet thuc!!!!";
14. return 1;
15. }
16. ...

* Lưu ý: hàm toNumber và HexaToDecimal chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

### Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân cải tiến

Một cách chuyển đổi khác:

* Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm thập phân.
* Đánh chỉ mục trong chuỗi thập lục phân:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| A | 7 | 0 | D |

* Chuyển đổi thập lục phân sang thập phân:

| Chỉ số | Ký tự | Số nguyên | Giá trị thập phân | Giải thích |
| --- | --- | --- | --- | --- |
|  |  |  | 0 | 0 |
| 0 | 'A' | 10 | 10 |  |
| 1 | '7' | 7 | 167 |  |
| 2 | '0' | 0 | 2.672 |  |
| 3 | 'D' | 13 | 42.765 |  |

Định nghĩa hàm chuyển đổi một chuỗi ký tự thập lục phân thành một giá trị trong hệ cơ số với điều kiện ràng buộc không đảo chuỗi nhị phân ban đầu.

1. long long HexaToDecimal(string n)
2. {
3. long long temp = 0;
4. for (int i = 0; i < n.length(); i++)
5. {
6. temp \*= 16;
7. temp += toNumber(n[i]);
8. }
9. return temp;
10. }

* Viết với ngôn ngữ .

1. Viết chương trình nhập một chuỗi thập lục phân và chuyển đổi chuỗi thập lục phân ra giá trị trong hệ thập phân mà không đảo chuỗi ban đầu.
2. #include <iostream>
3. using namespace std;
4. int toNumber(char);
5. long long HexaToDecimal(string);
6. int main()
7. {
8. string sHexa;
9. cout << "Nhap chuoi thap luc phan: ";
10. cin >> sHexa;
11. cout << "\nGia tri thap phan cua chuoi ";
12. cout << sHexa << " la: ";
13. cout << HexaToDecimal(sHexa);
14. cout << "\n\n\nKet thuc!!!!";
15. return 1;
16. }
17. ...

* Lưu ý: hàm toNumber và HexaToDecimal chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

### Chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân

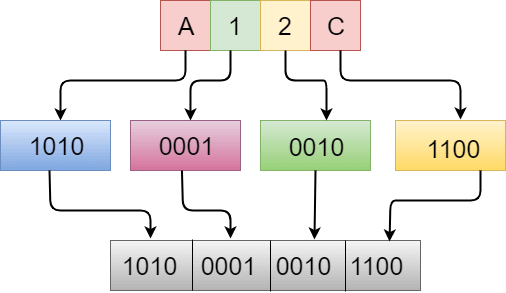
Bảng chuyển đổi ký tự số trong hệ hexa thành số nguyên.

| Ký tự số | Số nguyên | Số nhị phân | Nhóm |
| --- | --- | --- | --- |
| '0' | 0 | 0 | 0000 |
| '1' | 1 | 1 | 0001 |
| '2' | 2 | 10 | 0010 |
| '3' | 3 | 11 | 0011 |
| '4' | 4 | 100 | 0100 |
| '5' | 5 | 101 | 0101 |
| '6' | 6 | 110 | 0110 |
| '7' | 7 | 111 | 0111 |
| '8' | 8 | 1000 | 1000 |
| '9' | 9 | 1001 | 1001 |
| 'A' or 'a' | 10 | 1010 | 1010 |
| 'B' or 'b' | 11 | 1011 | 1011 |
| 'C' or 'c' | 12 | 1100 | 1100 |
| 'D' or 'd' | 13 | 1101 | 1101 |
| 'E' or 'e' | 14 | 1110 | 1110 |
| 'F' or 'f' | 15 | 1111 | 1111 |

Ví dụ chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân.

* Dữ liệu vào: 1AC5
* Dữ liệu ra: 0001.1010.1100.0101
* Giải thích:
  + Giá trị nhị phân của 1 là 0001.
  + Giá trị nhị phân của A là 1010.
  + Giá trị nhị phân của C là 1100.
  + Giá trị nhị phân của 5 là 0101.

Hình ảnh minh họa.



Khai báo hàm chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân

1. string HexaToBinary(string);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm HexaToBinary ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm HexaToBinary trả về một chuỗi ký tự.

Định nghĩa hàm chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân.

1. string HexaToBinary\_1(string n)
2. {
3. string temp;
4. for (int i = 0; i < n.size(); i++)
5. {
6. int index = toNumber(n[i]);
7. string group = DecimalToBinary(index);
8. while (group.size() < 4)
9. group = '0' + group;
10. temp = temp + group;
11. }
12. while (temp.size() > 1 && temp[0] == '0')
13. temp.erase(0, 1);
14. return temp;
15. }

* Viết với ngôn ngữ .

Định nghĩa hàm chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân.

1. string HexaToBinary\_2(string n)
2. {
3. string group[] = {"0000","0001","0010",
4. "0011","0100","0101","0110","0111",
5. "1000","1001","1010","1011","1100",
6. "1101","1110","1111"};
7. string temp;
8. for (int i = 0; i < n.size(); i++)
9. {
10. int index = toNumber(n[i]);
11. temp = temp + group[index];
12. }
13. while (temp.size() > 1 && temp[0] == '0')
14. temp.erase(0, 1);
15. return temp;
16. }

* Viết với ngôn ngữ .

Định nghĩa hàm chuyển đổi giá trị từ hệ đếm thập lục phân sang hệ đếm nhị phân.

1. string HexaToBinary\_3(string n)
2. {
3. unordered\_map<char, string> hexbinary
4. {
5. {'0',"0000"},{'1',"0001"},{'2',"0010"},
6. {'3',"0011"},{'4',"0100"},{'5',"0101"},
7. {'6',"0110"},{'7',"0111"},{'8',"1000"},
8. {'9',"1001"},{'A',"1010"},{'B',"1011"},
9. {'C',"1100"},{'D',"1101"},{'E',"1110"},
10. {'F',"1111"}
11. };
12. string temp;
13. for (int i = 0; i < n.size(); i++)
14. temp += hexbinary[toupper(n[i])];
15. while (temp.size() > 1 && temp[0] == '0')
16. temp.erase(0, 1);
17. return temp;
18. }

* Viết với ngôn ngữ .

1. Viết chương trình nhập một chuỗi thập lục phân và chuyển đổi chuỗi thập lục phân sang chuỗi nhị phân.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. int toNumber(char);
4. string HexaToBinary(string);
5. int main()
6. {
7. string sHexa;
8. cout << "Nhap chuoi thap luc phan: ";
9. cin >> sHexa;
10. cout << "\nBieu dien chuoi thap luc phan ";
11. cout << sHexa << " o he co so 2 la: ";
12. cout << HexaToBinary(sHexa);
13. cout << "\n\n\nKet thuc!!!!";
14. return 1;
15. }
16. ...

* Lưu ý: hàm toNumber và HexaToBinary chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

### Chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập lục phân

Bảng chuyển đổi ký tự số trong hệ hexa thành số nguyên.

| Ký tự số | Số nguyên | Số nhị phân | Nhóm |
| --- | --- | --- | --- |
| '0' | 0 | 0 | 0000 |
| '1' | 1 | 1 | 0001 |
| '2' | 2 | 10 | 0010 |
| '3' | 3 | 11 | 0011 |
| '4' | 4 | 100 | 0100 |
| '5' | 5 | 101 | 0101 |
| '6' | 6 | 110 | 0110 |
| '7' | 7 | 111 | 0111 |
| '8' | 8 | 1000 | 1000 |
| '9' | 9 | 1001 | 1001 |
| 'A' or 'a' | 10 | 1010 | 1010 |
| 'B' or 'b' | 11 | 1011 | 1011 |
| 'C' or 'c' | 12 | 1100 | 1100 |
| 'D' or 'd' | 13 | 1101 | 1101 |
| 'E' or 'e' | 14 | 1110 | 1110 |
| 'F' or 'f' | 15 | 1111 | 1111 |

Ví dụ chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập lục phân.

* Dữ liệu vào: 110001110
* Dữ liệu ra: 18E
* Giải thích:
  + Giá trị nhị phân: 110001110
  + Nhóm 4 bit từ phải sang: 1.1000.1110
  + Nhóm bit 0 vào đầu: 0001.1000.1110
  + Thực hiện chuyển đổi theo từng nhóm 4 bit: 18E
* Hình vẽ minh họa.

A picture containing text, screenshot, font, number

Description automatically generated

Khai báo hàm chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập lục phân.

1. string BinaryToHexa(string);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm BinaryToHexa ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm BinaryToHexa trả về một chuỗi ký tự.

Định nghĩa hàm chuyển đổi giá trị từ hệ đếm nhị phân sang hệ đếm thập lục phân.

1. string BinaryToHexa(string n)
2. {
3. while (n.size() % 4 != 0) n = '0' + n;
4. string temp;
5. for (int i = 0; i < n.size(); i += 4)
6. {
7. string group = n.substr(i, 4);
8. int index = BinaryToDecimal(group);
9. temp = temp + toChar(index);
10. }
11. return temp;
12. }

* Viết với ngôn ngữ .

1. Viết chương trình nhập một chuỗi nhị phân và chuyển đổi chuỗi nhị phân sang chuỗi thập lục phân.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. int toNumber(char);
4. char toChar(int);
5. long long BinaryToDecimal(string);
6. string BinaryToHexa(string);
7. int main()
8. {
9. string sBinary;
10. cout << "Nhap chuoi nhi phan: ";
11. cin >> sBinary;
12. cout << "\nBieu dien chuoi nhi phan ";
13. cout << sBinary << " o he co so 16 la: ";
14. cout << BinaryToHexa(sBinary);
15. cout << "\n\n\nKet thuc!!!!";
16. return 1;
17. }
18. ...

* Lưu ý: hàm toNumber, toChar, BinaryToDecimal và BinaryToHexa chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

## Hệ đếm tổng quát

### Khái niệm hệ đếm tổng quát

Hệ đếm (*numeral system*) là một hệ thống dùng để thể hiện các chữ số. Nói một cách khác, hệ đếm là một hệ thống các ký hiệu toán học để thể hiện các số của một tập hợp số, bằng cách sử dụng các chữ số hoặc các ký hiệu một cách nhất quán.

* Ví dụ: biểu tượng để được hiểu là biểu tượng hệ nhị phân của số hoặc biểu tượng hệ thập phân của số .
* Lý tưởng nhất, một hệ đếm sẽ:
* Thể hiện một tập hợp hữu ích của các số (ví dụ như các số nguyên, hoặc số hữu tỉ)
* Cung cấp cho mỗi số một biểu hiện duy nhất (hoặc ít nhất một cách trình bày quy chuẩn)
* Phản ánh các cấu trúc đại số và số học của các số.

### Hệ đếm cơ số b

* Trong hệ đếm cơ số , các kí hiệu được dùng có các giá trị tương ứng . Giả sử có biểu diễn:
* Trong đó số các chữ số bên trái, là số các chữ số bên phải dấu phân chia phần nguyên và phần phân của số và các phải thoả mãn điều kiện:
* Khi đó giá trị của số được tính theo công thức:
* Chú ý để phân biệt số được biểu diễn ở hệ đếm nào người ta viết cơ số làm chỉ số dưới của số đó. Ví dụ: là biểu diễn ở hệ đếm .

### Các hệ đếm phổ biến

* Hệ đếm thập phân.
* Hệ đếm nhị phân.
* Hệ đếm thập lục phân.
* Hệ đếm bát phân.

### Chuyển đổi giá trị trong biểu diễn cơ số b sang cơ số 10

* Viết với ngôn ngữ .
* Viết với ngôn ngữ .

### Chuyển đổi giá trị trong biểu diễn cơ số 10 sang cơ số b

* Viết với ngôn ngữ .
* Viết với ngôn ngữ .

### Chuyển đổi giá trị trong biểu diễn cơ số b sang cơ số c

* Viết với ngôn ngữ .
* Viết với ngôn ngữ .

## Bài tập hệ đếm

1. Cho một số nguyên bit có dấu , tìm giá trị đảo ngược của . Trong tình huống giá trị đảo ngược của khiến giá trị nằm ngoài phạm vi số nguyên bit có dấu thì trả về 0.

Các thông số của bài toán:

* Giới hạn:
* Dữ liệu vào:
  + Số
* Dữ liệu ra:
  + Giá trị đảo ngược của .

Định nghĩa hàm:

1. int Reverse(int n)
2. {
3. int dn = 0;
4. while (n)
5. {
6. if (dn > INT\_MAX/10 || dn < INT\_MIN/10)
7. return 0;
8. dn = dn \* 10 + n % 10;
9. n /= 10;
10. }
11. return dn;
12. }

* Viết với ngôn ngữ .

1. Cho một số nguyên bit có dấu , kiểm tra số nguyên có đối xứng hay không?

Các thông số của bài toán:

* Giới hạn:
* Dữ liệu vào:
  + Số
* Dữ liệu ra:
  + hoặc .
* Dữ liệu kiểm thử 1:
  + Dữ liệu vào:
  + Dữ liệu ra:
    - .
  + Giải thích:
* Dữ liệu kiểm thử 2:
  + Dữ liệu vào:
  + Dữ liệu ra:
    - .
  + Giải thích
* Dữ liệu kiểm thử 3:
  + Dữ liệu vào:
  + Dữ liệu ra:
    - .
  + Giải thích

Định nghĩa hàm:

1. bool isPalindrome(int n)
2. {
3. if (n < 0)
4. return false;
5. int nn = n;
6. int dn = 0;
7. while (nn)
8. {
9. if (dn > INT\_MAX / 10)
10. return false;
11. dn = dn \* 10 + nn % 10;
12. nn /= 10;
13. }
14. return (dn == n);
15. }

* Viết với ngôn ngữ .

1. Cho một số nguyên , hãy chuyển nó thành số La Mã.

* Số La Mã được biểu diễn bằng 7 kí tự sau: và .

|  |  |
| --- | --- |
| Ký hiệu | Giá trị |
| I | 1 |
| V | 5 |
| X | 10 |
| L | 50 |
| C | 100 |
| D | 500 |
| M | 1.000 |

* Ví dụ,
* 2 được viết như là II bên số La Mã, chỉ là 2 số 1 cộng lại với nhau.
* 12 được viết như là XII, đơn giản là kết hợp giữa X và II.
* Số 27 được viết là XXVII, là sự kết hợp giữa XX + V + II
* Số La Mã thường được viết theo từ cao nhất đến thấp nhất từ trái qua phải.
* Tuy nhiên, số La Mã của 4 không phải là IIII. Thay vào đó, số La Mã của 4 được viết là IV. Bởi vì số 1 đằng trước số 5 như một phép trừ để chúng ta có được số 4. Quy luật này cũng được áp dụng tương tự cho số 9, được viết như là IX. Sau đây là 6 trường hợp mà phép trừ được sử dụng:
* I có thể được đặt trước V (5) và X (10) để tạo số 4 và 9
* X có thể được đặt trước L (50) và C (100) để tạo số 40 và 90
* C có thể được đặt trước D (500) và M(1000) để tạo số 400 và 900.

Các thông số của bài toán:

* Giới hạn:
* Dữ liệu vào:
* Dữ liệu ra:
* Trường hợp kiểm thử 1:
  + Dữ liệu vào: .
  + Dữ liệu ra: “III”
  + Giải thích: 3 được viết bằng ba số 1 bên số La Mã
* Ví dụ 2:
  + Dữ liệu vào: .
  + Dữ liệu ra: “LVIII”
  + Giải thích: L = 50, V = 5, III = 3
* Ví dụ 3:
  + Dữ liệu vào: .
  + Dữ liệu ra: “MCMXCIV”
  + Giải thích: M = 1000, CM = 900, XC = 90 và IV = 4
* Ở một góc nhìn đơn giản chữ số La Mã được biểu diễn bởi 13 cụm ký tự sau:
* Số La Mã được biểu diễn bằng 7 kí tự sau: và .

|  |  |  |
| --- | --- | --- |
| STT | Ký hiệu | Giá trị |
| 1 | I | 1 |
| 2 | IV | 4 |
| 3 | V | 5 |
| 4 | IX | 9 |
| 5 | X | 10 |
| 6 | XL | 40 |
| 7 | L | 50 |
| 8 | XC | 90 |
| 9 | C | 100 |
| 10 | CD | 400 |
| 11 | D | 500 |
| 12 | CM | 900 |
| 13 | M | 1.000 |

Định nghĩa hàm:

1. string intToRoman(int n)
2. {
3. string temp;
4. int num[] = { 1,4,5,9,10,40,50,90,100,400,
5. 500,900,1000 };
6. string sym[] = { "I","IV","V","IX","X","XL",
7. "L","XC","C","CD","D","CM","M" };
8. int i = 12;
9. while (n > 0)
10. {
11. int div = n / num[i];
12. while (div--)
13. temp = temp + sym[i];
14. n = n % num[i];
15. i--;
16. }
17. return temp;
18. }

* Viết với ngôn ngữ .

1. Gọi là một số có chữ số. Hãy lập chương trình tìm tất cả các số có chữ số thỏa mãn biểu thức: .
2. Tìm số chính phương lớn nhất của kiểu dữ liệu unsigned long long.
3. Tìm số nguyên tố lớn nhất của kiểu dữ liệu unsigned long long.
4. Tìm giá trị lớn nhất sao cho biểu thức sau đúng (hai mươi chữ số).

# SỐ NGUYÊN TỐ – PRIME NUMBER

## Khái niệm số nguyên tố

Số nguyên tố (*prime numbers*) là số nguyên dương lớn hơn và chỉ có đúng hai ước số.

## Khái niệm hợp số

Hợp số (*composite number*) là các số nguyên dương lớn hơn và có nhiều hơn hai ước số.

## Kiếm tra tính nguyên tố chính xác

Khai báo hàm kiểm tra một số nguyên có phải là số nguyên tố hay không.

1. bool ktNguyenTo(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm ktNguyenTo ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm ktNguyenTo trả về một trong hai giá trị.
* Giá trị : đối số đầu vào là số nguyên tố.
* Giá trị : đối số đầu vào không là số nguyên tố.

### Giải thuật kiểm tra nguyên tố theo định nghĩa

Ý tưởng giải thuật kiểm tra số nguyên tố theo định nghĩa:

* Duyệt qua các số nguyên từ tới và đếm số lượng ước số của .
* Nếu số lượng ước số của bằng thì kết luận là số nguyên tố (trả về giá trị ).
* Ngược lại, không là số nguyên tố (trả về giá trị ).

Định nghĩa hàm kiểm tra một số nguyên có phải là số nguyên tố hay không.

1. bool ktNguyenTo(long long n)
2. {
3. int dem = 0;
4. for (long long i = 1; i <= n; i++)
5. if (n % i == 0)
6. dem++;
7. if (dem == 2)
8. return true;
9. return false;
10. }

* Viết với ngôn ngữ .
* Viết lại hàm trên gọn hơn:

1. bool ktNguyenTo(long long n)
2. {
3. int dem = 0;
4. for (long long i = 1; i <= n; i++)
5. if (n % i == 0)
6. dem++;
7. return (dem == 2);
8. }

* Viết với ngôn ngữ .

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

### Giải thuật kiểm tra số nguyên tố cải tiến

Ý tưởng cải tiến rất đơn giản:

* Số là nguyên tố thì có đúng ước số là và .
* Ta duyệt qua tất cả các số nguyên từ tới , nếu như có số nào là ước của thì kết luận không phải số nguyên tố.
* Nếu duyệt xong mà không có số nào là ước của thì kết luận là số nguyên tố.

Định nghĩa hàm:

1. bool ktNguyenTo(long long n)
2. {
3. if (n <= 1)
4. return false;
5. for (long long i = 2; i <= n-1; i++)
6. if (n % i == 0)
7. return false;
8. return true;
9. }

* Viết với ngôn ngữ .

Hàm cài đặt trên vẫn có độ phức tạp .

* Định nghĩa hàm trên có thể được viết lại gọn hơn:

1. bool ktNguyenTo(long long n)
2. {
3. for (long long i = 2; i < n; i++)
4. if (n % i == 0)
5. return 0;
6. return (n>1);
7. }

* Viết với ngôn ngữ .
* Nhận xét:
* Giải thuật cải tiến này nhanh hơn giải thuật kiểm tra nguyên tố theo định nghĩa trong trường hợp là hợp số.
* Tuy nhiên, khi là nguyên tố số lần lặp của giải thuật vẫn tiệm cận . Do đó, giải thuật vẫn có độ phức tạp .

Định nghĩa hàm trên có thể được viết lại tốt hơn dựa vào nhận xét ngoài ước số , số nguyên dương chắc chắn không có ước số lớn hơn .

1. bool ktNguyenTo(long long n)
2. {
3. for (long long i = 2; i <= n/2; i++)
4. if (n % i == 0)
5. return 0;
6. return (n>1);
7. }

* Viết với ngôn ngữ .
* Nhận xét: giải thuật cải tiến này nhanh hơn giải thuật trên 2 lần. Tuy nhiên, giải thuật vẫn có độ phức tạp .

### Cải tiến dựa vào định lý toán học

Định lý: mọi hợp số (*composite number*) luôn luôn có ước nguyên tố nhỏ hơn hay bằng căn bậc hai.

* Chứng minh:
* Gọi hợp số là .
* Ta có thể viết hợp số dưới dạng:
* với .
* Do:
* Nên:
* Suy ra:
* Kết luận: mọi hợp số luôn luôn có ước nguyên tố nhỏ hơn hay bằng căn bậc hai của nó.

Từ kết quả của định lý trên, việc kiểm tra với từ đến là không cần thiết, mà chỉ cần kiểm tra từ đến .

* Định nghĩa hàm:

1. bool ktNguyenTo(long long n)
2. {
3. if (n <= 1)
4. return false;
5. for (long long i = 2; i <= sqrt(n); i++)
6. if (n % i == 0)
7. return false;
8. return true;
9. }

* Viết với ngôn ngữ .

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Trong tình huống xấp xỉ (một triệu) thì hàm ktNguyenTo chỉ chạy khoảng lần

### Cải tiến dựa vào nhận xét logic kiểu của dân tin học

Hàm cài đặt trên ta có thể thay điều kiện lặp thành   
 vì chi phí tính căn trội hơn chi phí thực hiện phép toán nhân.

* Định nghĩa hàm:

1. bool ktNguyenTo(long long n)
2. {
3. if (n <= 1)
4. return false;
5. for (long long i = 2; i\*i <= n; i++)
6. if (n % i == 0)
7. return false;
8. return true;
9. }

* Viết với ngôn ngữ .
* Hàm cài đặt trên vẫn có độ phức tạp: do ta phải duyệt hết các số từ đến .

Hàm cài đặt trên có thể cải tiến để tốc độ tăng gấp lần với một số nhận xét sau:

* Số nguyên tố chẵn duy nhất là .
* Các số chẵn khác không là số nguyên tố.
* Biến chạy từ với bước nhảy là (biến không duyệt các số chẵn nữa).
* Hàm cài đặt cải tiến:

1. bool ktNguyenTo(long long n)
2. {
3. if (n <= 1)
4. return false;
5. if (n == 2)
6. return true;
7. if (n % 2 == 0)
8. return false;
9. for (long long i = 3; i \* i <= n; i += 2)
10. if (n % i == 0)
11. return false;
12. return true;
13. }

* Viết với ngôn ngữ .

Tuy tốc độ thực hiện của hàm cài đặt tăng gấp hai lần nhưng giải thuật vẫn có độ phức tạp: .

* Lưu ý: một số nguyên bất kỳ có thể biểu diễn dưới một trong bảy dạng sau:
* .
* Các ví dụ:
* Ví dụ 01: .
* Ví dụ 02: .
* Ví dụ 03: .
* Ví dụ 04: .
* Ví dụ 05: .
* Ví dụ 06: .

Hàm trên có thể tiếp tục cải tiến để tốc độ nhanh hơn với nhận xét sau: trừ số và số các số nguyên tố đều có dạng vì các số có dạng:

* chia hết cho 2,
* chia hết cho 2 và
* chia hết cho .
* Hàm cài đặt cải tiến:

1. bool ktNguyenTo(long long n)
2. {
3. if (n <= 1)
4. return false;
5. if (n == 2)
6. return true;
7. if (n == 3)
8. return true;
9. if (n % 2 == 0)
10. return false;
11. if (n % 3 == 0)
12. return false;
13. for (long long i = 5; i \* i <= n; i += 6)
14. if ((n % i == 0) || (n % (i + 2) == 0))
15. return false;
16. return true;
17. }

* Viết với ngôn ngữ .

Tuy tốc độ thực hiện có tăng hơn gấp đôi hàm cài đặt trên nhưng giải thuật vẫn có độ phức tạp: .

1. Viết chương trình nhập một số nguyên kiểu long long. Kiểm tra số nguyên đó có phải là số nguyên tố hay không.

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. bool isPrime(long long);
4. int main()
5. {
6. long long n;
7. cout << "Nhap n: ";
8. cin >> n;
9. if (isPrime(n))
10. cout << n << " la so nguyen to!!!";
11. else
12. cout << n << " khong la so nguyen to!!!";
14. cout << "\n\n\nKet thuc!!!!";
15. return 1;
16. }
17. ...

* Lưu ý: hàm isPrime chúng ta tham khảo bên trên.
* Viết với ngôn ngữ .

## Kiếm tra tính nguyên tố theo xác suất

### Phương pháp Fermat

Định lý Fermat nhỏ: nếu là một số nguyên tố thì với mọi số nguyên tố cùng nhau với , ta luôn có hoặc a ^ .

* Từ định lý này, nếu ta muốn kiểm tra số có là nguyên tố không, ta lấy ngẫu nhiên các số và kiểm tra xem đẳng thức trên có đúng không.
* Nếu đẳng thức không đúng với một giá trị nào đó thì là hợp số.
* Nếu đẳng thức đúng với nhiều giá trị của , ta có thể nói rằng "có khả năng cao" là số nguyên tố.
* Tất nhiên là chúng ta sẽ tìm cách tăng "khả năng" này lên bằng cách thực hiện nhiều lần kiểm tra với các giá trị khác nhau.
* Hàm cài đặt.
* Viết với ngôn ngữ .

### Phương pháp Miller – Rabin

* Viết với ngôn ngữ .
* Viết với ngôn ngữ .

### Phương pháp Solovay – Strassen

Kiểm tra Solovay-Strassen là một trong các phương pháp kiểm tra tính nguyên tố theo xác suất do Robert M. Solovay và Volker Strassen phát triển.

* Viết với ngôn ngữ .

#### Ký hiệu Legendre

* Legendre đưa ra ký hiệu mang tên ông cho số nguyên tố lẻ và số nguyên :
* Trong đó: là một bình phương đúng modulo nghĩa là tồn tại số nguyên sao cho .
* Ví dụ:
* vì chia hết .
* vì chia hết .
* vì tồn tại và và .

#### Tiêu chuẩn Euler

* Euler chứng minh rằng với mọi số nguyên tố và số , , .

#### Ký hiệu Jacobi

* Ký hiệu Jacobi là mở rộng của Ký hiệu Legendre cho số tự nhiên lẻ .
* Giả sử:
* là dạng phân tích tiêu chuẩn của và
* là số nguyên bất kỳ
* Ký hiệu Jacobi:

## Tìm các số nguyên tố trong đoạn

* Các số nguyên tố nhỏ là .
* Cách tiếp cận ngây thơ là thử lần lượt các số nguyên trong đoạn rồi kiểm tra tính nguyên tố của .

Khai báo hàm tìm các số nguyên tố trong đoạn :

1. bool isPrime(long long);
2. vector<long long> findPrime(long long);

Định nghĩa hàm tìm các số nguyên tố trong đoạn :

1. vector<long long> findPrime(long long n)
2. {
3. vector<long long> prime;
4. for (long long p = 2; p <= n; p++)
5. if (isPrime(p))
6. prime.push\_back(p);
7. return prime;
8. }

* Viết với ngôn ngữ .

1. Viết chương trình nhập vào số nguyên . Tìm và xuất các số nguyên tố trong đoạn .

* Viết với ngôn ngữ .

1. #include <iostream>
2. #include <string>
3. #include <vector>
4. using namespace std;
5. bool isPrime(long long);
6. string outVector(vector<long long>);
7. vector<long long> findPrime(long long);
8. int main()
9. {
10. long long n;
11. cout << "Nhap n (n <= 10^9): ";
12. cin >> n;
13. cout << "\nCac so nguyen to nho hon bang ";
14. cout << n << " la: ";
15. cout << outVector(findPrime(n));
16. cout << "\n\n\nKet thuc!!!!";
17. return 1;
18. }
19. string outVector(vector<long long> a)
20. {
21. string temp;
22. long long i = 0;
23. while (i < a.size())
24. {
25. temp = temp + " " + to\_string(a[i]);
26. i++;
27. }
28. return temp;
29. }
30. ...

* Lưu ý: hàm isPrime và findPrime chúng ta tham khảo ở các trang trước.
* Viết với ngôn ngữ .

## Thuật toán sàng Eratosthene – Sieve of Eratosthenes

* Thuật toán sàng là một giải thuật cổ xưa do nhà Toán học người Hy Lạp phát minh.
* Tương truyền, khi tìm ra thuật toán, nhà khoa học đã lấy lá cọ và ghi tất cả các số từ cho đến lên đó, sau đó chọc thủng các hợp số và giữ nguyên các số nguyên tố.
* Bảng số nguyên tố còn lại trông rất giống một cái sàng. Do đó, thuật toán có tên gọi là sàng .

### Minh họa từng bước thuật toán Eratosthenes

Tìm tất cả các số nguyên tố trong đoạn .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số không là số nguyên tố.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của (rất tiếc là không loại bỏ được bội số nào).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Các số chưa bị đánh dấu đều là nguyên tố.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Kết luận: các số nguyên tố nhỏ 100 là .

### Thuật toán sàng Eratosthenes

Bài toán: tìm các số nguyên tố trong đoạn bằng thuật toán sàng .

* Thuật toán sàng :
* Xét các số nguyên tố từ tới , với mỗi số nguyên tố ta sẽ đánh dấu các bội của mà lớn hơn đều là hợp số.
* Sau khi duyệt các số nguyên tố từ tới xong, tất cả các số trong đoạn chưa được đánh dấu sẽ là số nguyên tố.

### Hàm cài đặt thuật toán sàng Eratosthenes

Bài toán: Tìm các số nguyên tố trong đoạn bằng thuật toán sàng Eratosthenes.

* Khai báo hàm:

1. vector<long long> Eratosthenes(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm Eratosthenes ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm Eratosthenes trả về một véc tơ chứa các số nguyên tố trong đoạn .
* Định nghĩa hàm:

1. vector<long long> Eratosthenes(long long n)
2. {
3. vector<bool> isprime(n + 1, true);
4. isprime[0] = false;
5. isprime[1] = false;
6. for (long long p = 2; p <= n; p++)
7. if (isprime[p] == true)
8. for (int i = 2 \* p; i <= n; i += p)
9. isprime[i] = false;
10. vector<long long> prime;
11. for (long long p = 2; p <= n; p++)
12. if (isprime[p] == true)
13. prime.push\_back(p);
14. return prime;
15. }

Sàng Eratosthenes thể hiện trong câu lệnh:

1. vector<bool> isprime(n + 1, true);

* Hàm trên có thể cải tiến bằng cách cho vòng lặp theo biến bắt đầu chạy từ .

1. for (int i = 2 \* p; i <= n; i += p)

Lí do vòng lặp theo duyệt các số nguyên tố tăng dần ( chạy bắt đầu từ tới ), khi tới số nguyên tố ( đã chạy từ ) thì các bội đều đã bị loại đi trước đó bởi các số nguyên tố nhỏ hơn rồi.

1. for (int i = p \* p; i <= n; i += p)

* Viết với ngôn ngữ .
* Định nghĩa hàm cải tiến:

1. vector<long long> Eratosthenes(long long n)
2. {
3. vector<bool> isprime(n + 1, true);
4. isprime[0] = false;
5. isprime[1] = false;
6. for (long long p = 2; p <= n; p++)
7. if (isprime[p] == true)
8. for (int i = p \* p; i <= n; i += p)
9. isprime[i] = false;
10. vector<long long> prime;
11. for (long long p = 2; p <= n; p++)
12. if (isprime[p] == true)
13. prime.push\_back(p);
14. return prime;
15. }

* Viết với ngôn ngữ .

Hàm trên có thể cải tiến bằng cách cho vòng lặp theo chạy từ 2 cho tới vì ta có định lý: mọi hợp số (*composite number*) luôn luôn có ước nguyên tố nhỏ hơn hay bằng căn bậc hai của nó.

1. vector<long long> Eratosthenes(long long n)
2. {
3. vector<bool> isprime(n + 1, true);
4. isprime[0] = false;
5. isprime[1] = false;
6. for (long long p = 2; p \* p <= n; p++)
7. if (isprime[p] == true)
8. for (int i = 2 \* p; i <= n; i += p)
9. isprime[i] = false;
10. vector<long long> prime;
11. for (long long p = 2; p <= n; p++)
12. if (isprime[p] == true)
13. prime.push\_back(p);
14. return prime;
15. }

* Viết với ngôn ngữ .
* Đánh giá độ phức tạp của thuật toán sàng :
* vòng lặp bên trong lặp lần.
* vòng lặp bên trong lặp lần.
* vòng lặp bên trong lặp lần.
* Tổng số lần lặp là: .

Kết luận: độ phức tạp của thuật toán là: .

* Độ phức tạp bộ nhớ của thuật toán sàng vì ta cần một bộ nhớ phụ có kích thước để đánh dấu các số nguyên thuộc đoạn là nguyên tố hay hợp số.
* Bộ nhớ phụ thể hiện trong câu lệnh:

1. vector<bool> isprime(n + 1, true);
2. Viết chương trình nhập vào số nguyên . Tìm và xuất các số nguyên tố trong đoạn bằng thuật toán sàng .

* Viết với ngôn ngữ .
* Viết với ngôn ngữ .

## Phân tích thừa số nguyên tố

Phân tích thừa số nguyên tố (*prime factorization*) là một chủ đề trong lập trình thi đấu (*competitive programming*).

Phân tích ra thừa số nguyên tố.

| Giá trị | Thừa số nguyên tố |
| --- | --- |
| 1.000 | 2 |
| 500 | 2 |
| 250 | 2 |
| 125 | 5 |
| 25 | 5 |
| 5 | 5 |
| 1 |  |

Vậy: .

### Giải thuật phân tích thừa số nguyên tố ngây thơ

Ý tưởng: ta xét mọi số nguyên tố bắt đầu từ , nếu chia hết cho số nguyên tố thì chia cho tới khi không thể chia hết nữa, rồi tăng lên và lặp lại công việc tới khi .

* Khai báo hàm:

1. vector<long long> Factorization(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm Factorization ta có giá trị của tham số thứ nhất chưa?
* Giá trị trả về: hàm Factorization trả về một véc tơ chứa các thừa số nguyên tố trong trường hợp giá trị của đối số đầu vào không thể phân tích thành thừa số nguyên tố thì hàm sẽ trả về véc tơ rỗng không chứa phần tử nào.
* Hàm cài đặt:

1. vector<long long> Factorization(long long n)
2. {
3. vector<long long> factorprime;
4. long long p = 2;
5. while (n > 1)
6. {
7. while (n % p == 0)
8. {
9. factorprime.push\_back(p);
10. n /= p;
11. }
12. p++;
13. }
14. return factorprime;
15. }

Chạy từng bước với

* Kết quả:

Chạy từng bước với

* Kết quả:

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến trong tình huống là số nguyên tố.

* Viết với ngôn ngữ .

### Cải tiến giải thuật phân tích thừa số nguyên tố ngây thơ

* Nhắc lại định lý: mọi hợp số (*composite number*) luôn luôn có ước nguyên tố nhỏ hơn hay bằng căn bậc hai của nó.

Từ định lý trên ta có nhận xét sau: không thể xảy ra trường hợp mọi thừa số nguyên tố của đều lớn hơn , do đó chúng ta chỉ cần xét các ước số của từ tới và chia dần cho các ước của nó tới khi bằng . Nếu không thể tìm được ước nào trong đoạn thì phải là một số nguyên tố.

* Hàm cài đặt:

1. vector<long long> Factorization(long long n)
2. {
3. vector<long long> factorprime;
4. long long p = 2;
5. while (n > 1 && p \* p <= n)
6. {
7. while (n % p == 0)
8. {
9. factorprime.push\_back(p);
10. n /= p;
11. }
12. p++;
13. }
14. if (n > 1)
15. factorprime.push\_back(n);
16. return factorprime;
17. }

* Viết với ngôn ngữ .

Chạy từng bước với

* Kết quả:

Chạy từng bước với

* Kết quả:

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến trong tình huống là số nguyên tố.

* Khi xuất các thừa số của cần xuất dạng lũy thừa (ví dụ . Chúng ta có thể tham khảo hàm sau:

1. string outPowerVector(vector<long long> a)
2. {
3. string temp;
4. long long i = 0;
5. while (i < a.size())
6. {
7. temp = temp + " " + to\_string(a[i]);
8. long long count = 1;
9. while (i + 1 < a.size() && a[i+1] == a[i])
10. {
11. count++;
12. i++;
13. }
14. if (count > 1)
15. temp = temp + "^" + to\_string(count);
16. i++;
17. }
18. return temp;
19. }

* Viết với ngôn ngữ .

### Phân tích thừa số nguyên tố với sàng Eratosthenes

* Từ hai giải thuật trên ta thấy ở mỗi bước phân tích thừa số nguyên tố ta cần tìm ra ước số nguyên tố nhỏ nhất của rồi chia cho ước đó.
* Ta sẽ cải tiến sàng Eratosthenes đi một chút để lấy được ngay ước nguyên tố nhỏ nhất của với độ phức tạp ở mỗi bước phân tích, điều này sẽ giúp giảm thời gian chạy đi đáng kể.

Tìm tất cả các số nguyên tố trong đoạn và các ước nguyên tố nhỏ nhất của tất cả các số nguyên trong đoạn .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số không là số nguyên tố.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

Ghi nhận các bội số của có ước số nguyên tố nhỏ nhất là .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 9 | 2 |
| 11 | 2 | 13 | 2 | 15 | 2 | 17 | 2 | 19 | 2 |
| 21 | 2 | 23 | 2 | 25 | 2 | 27 | 2 | 29 | 2 |
| 31 | 2 | 33 | 2 | 35 | 2 | 37 | 2 | 39 | 2 |
| 41 | 2 | 43 | 2 | 45 | 2 | 47 | 2 | 49 | 2 |
| 51 | 2 | 53 | 2 | 55 | 2 | 57 | 2 | 59 | 2 |
| 61 | 2 | 63 | 2 | 65 | 2 | 67 | 2 | 69 | 2 |
| 71 | 2 | 73 | 2 | 75 | 2 | 77 | 2 | 79 | 2 |
| 81 | 2 | 83 | 2 | 85 | 2 | 87 | 2 | 89 | 2 |
| 91 | 2 | 93 | 2 | 95 | 2 | 97 | 2 | 99 | 2 |

* Thống kê có số chẵn trong đoạn có ước nguyên tố nhỏ nhất là .

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 9 | 2 |
| 11 | 2 | 13 | 2 | 15 | 2 | 17 | 2 | 19 | 2 |
| 21 | 2 | 23 | 2 | 25 | 2 | 27 | 2 | 29 | 2 |
| 31 | 2 | 33 | 2 | 35 | 2 | 37 | 2 | 39 | 2 |
| 41 | 2 | 43 | 2 | 45 | 2 | 47 | 2 | 49 | 2 |
| 51 | 2 | 53 | 2 | 55 | 2 | 57 | 2 | 59 | 2 |
| 61 | 2 | 63 | 2 | 65 | 2 | 67 | 2 | 69 | 2 |
| 71 | 2 | 73 | 2 | 75 | 2 | 77 | 2 | 79 | 2 |
| 81 | 2 | 83 | 2 | 85 | 2 | 87 | 2 | 89 | 2 |
| 91 | 2 | 93 | 2 | 95 | 2 | 97 | 2 | 99 | 2 |

Ghi nhận các bội số của có ước số nguyên tố nhỏ nhất là .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 25 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 35 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 49 | 2 |
| 3 | 2 | 53 | 2 | 55 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 65 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 77 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 85 | 2 | 3 | 2 | 89 | 2 |
| 91 | 2 | 3 | 2 | 95 | 2 | 97 | 2 | 3 | 2 |

* Thống kê có số chia hết cho trong đoạn có ước nguyên tố nhỏ nhất là .

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 25 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 35 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 49 | 2 |
| 3 | 2 | 53 | 2 | 55 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 65 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 77 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 85 | 2 | 3 | 2 | 89 | 2 |
| 91 | 2 | 3 | 2 | 95 | 2 | 97 | 2 | 3 | 2 |

Ghi nhận các bội số của có ước số nguyên tố nhỏ nhất là .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 5 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 5 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 49 | 2 |
| 3 | 2 | 53 | 2 | 5 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 5 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 77 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 5 | 2 | 3 | 2 | 89 | 2 |
| 91 | 2 | 3 | 2 | 5 | 2 | 97 | 2 | 3 | 2 |

* Thống kê có số chia hết cho trong đoạn có ước nguyên tố nhỏ nhất là .

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 5 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 5 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 49 | 2 |
| 3 | 2 | 53 | 2 | 5 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 5 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 77 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 5 | 2 | 3 | 2 | 89 | 2 |
| 91 | 2 | 3 | 2 | 5 | 2 | 97 | 2 | 3 | 2 |

Ghi nhận các bội số của có ước số nguyên tố nhỏ nhất là .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 5 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 5 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 7 | 2 |
| 3 | 2 | 53 | 2 | 5 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 5 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 7 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 5 | 2 | 3 | 2 | 89 | 2 |
| 7 | 2 | 3 | 2 | 5 | 2 | 97 | 2 | 3 | 2 |

* Thống kê có số chia hết cho trong đoạn có ước nguyên tố nhỏ nhất là .

Số là số nguyên tố. Đánh dấu và loại bỏ các bội số của (rất tiếc là không loại bỏ được bội số nào).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 5 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 5 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 7 | 2 |
| 3 | 2 | 53 | 2 | 5 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 5 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 7 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 5 | 2 | 3 | 2 | 89 | 2 |
| 7 | 2 | 3 | 2 | 5 | 2 | 97 | 2 | 3 | 2 |

Các số chưa bị đánh dấu đều là nguyên tố.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 5 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 5 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 7 | 2 |
| 3 | 2 | 53 | 2 | 5 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 5 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 7 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 5 | 2 | 3 | 2 | 89 | 2 |
| 7 | 2 | 3 | 2 | 5 | 2 | 97 | 2 | 3 | 2 |

Nhắc lại hàm cài đặt thuật toán sàng .

1. vector<long long> Eratosthenes(long long n)
2. {
3. vector<bool> isprime(n + 1, true);
4. isprime[0] = false;
5. isprime[1] = false;
6. for (long long p = 2; p \* p <= n; p++)
7. if (isprime[p] == true)
8. for (int i = 2 \* p; i <= n; i += p)
9. isprime[i] = false;
10. vector<long long> prime;
11. for (long long p = 2; p <= n; p++)
12. if (isprime[p] == true)
13. prime.push\_back(p);
14. return prime;
15. }

* Giải thuật phân tích thừa số nguyên tố với sàng :
* Sử dụng thuật toán sàng tìm ước nguyên tố nhỏ nhất của mọi số nguyên trong đoạn .
* Chia cho ước nguyên tố nhỏ nhất của cho tới khi .
* Hàm cài đặt phân tích thừa số nguyên tố với sàng :

1. vector<long long> Factorization(long long n)
2. {
3. vector<long long> minprime(n + 1);
4. for (long long p = 0; p <= n; p++)
5. minprime[p] = p;
6. for (long long p = 2; p \* p <= n; p++)
7. if (minprime[p] == p)
8. for (int i = p \* p; i <= n; i += p)
9. if (minprime[i] == i)
10. minprime[i] = p;
11. vector<long long> factorprime;
12. while (n > 1)
13. {
14. factorprime.push\_back(minprime[n]);
15. n /= minprime[n];
16. }
17. return factorprime;
18. }

Giải thuật có độ phức tạp: do thuật toán phải xây dựng sàng . Tuy nhiên, thao tác phân tích một số thành thừa số nguyên tố chỉ mất độ phức tạp . Điều này sẽ rất có lợi trong các bài toán phải phân tích thừa số nguyên tố nhiều lần.

* Viết với ngôn ngữ .
* Chạy từng bước phân tích ra thừa số nguyên tố.
* Sử dụng thuật toán sàng tìm ước nguyên tố nhỏ nhất của mọi số nguyên trong đoạn .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 2 | 5 | 2 | 7 | 2 | 3 | 2 |
| 11 | 2 | 13 | 2 | 3 | 2 | 17 | 2 | 19 | 2 |
| 3 | 2 | 23 | 2 | 5 | 2 | 3 | 2 | 29 | 2 |
| 31 | 2 | 3 | 2 | 5 | 2 | 37 | 2 | 3 | 2 |
| 41 | 2 | 43 | 2 | 3 | 2 | 47 | 2 | 7 | 2 |
| 3 | 2 | 53 | 2 | 5 | 2 | 3 | 2 | 59 | 2 |
| 61 | 2 | 3 | 2 | 5 | 2 | 67 | 2 | 3 | 2 |
| 71 | 2 | 73 | 2 | 3 | 2 | 7 | 2 | 79 | 2 |
| 3 | 2 | 83 | 2 | 5 | 2 | 3 | 2 | 89 | 2 |
| 7 | 2 | 3 | 2 | 5 | 2 | 97 | 2 | 3 | 2 |

* Kết quả:

## Sàng Eratosthenes trên đoạn

Bài toán: tìm các số nguyên tố trong đoạn bằng thuật toán sàng .

* Giải thuật sàng trên đoạn
* Sàng lọc trước một mảng gồm các số nguyên tố trong đoạn .
* Xét các số nguyên tố từ tới , với mỗi số nguyên tố ta sẽ đánh dấu các bội của mà lớn hơn nằm trong đoạn .
* Sau khi duyệt xong, tất cả các số chưa được đánh dấu trong đoạn sẽ là số nguyên tố.
* Khai báo hàm:
* Viết với ngôn ngữ .
* Viết với ngôn ngữ .

## Định lý Wilson

Định lý. Cho là số tự nhiên lớn hơn , khi đó là số nguyên tố khi và chỉ khi chia hết cho .

## Bài tập số nguyên tố

1. Định nghĩa hàm tìm ước nguyên tố nhỏ nhất của các số nguyên trong đoạn bằng thuật toán sàng .
2. Mọi số tự nhiên đều có thể viết được dưới dạng tổng của hai số nguyên tố. Định nghĩa hàm tách một số tự nhiên thành tổng của hai số nguyên tố.
3. Định nghĩa hàm tìm ước nguyên tố lớn nhất của các số nguyên trong đoạn bằng thuật toán sàng .
4. Tính tổng các thừa số nguyên tố của số nguyên dương .

* Ví dụ:
* Dữ liệu vào: .
* Dữ liệu ra: .
* Giải thích: .
* Tổng các thừa số nguyên tố: .

1. Tính tổng các thừa số nguyên tố khác nhau của số nguyên dương .

* Ví dụ:
* Dữ liệu vào: .
* Dữ liệu ra: .
* Giải thích: .
* Tổng các thừa số nguyên tố khác nhau: .

1. Goldbach's Conjecture (UVa 543) Cho số nguyên dương , tìm cặp số nguyên tố mà tổng của chúng bằng và chênh lệch giữa số là lớn nhất.

* Hướng dẫn giải đề nghị:
* Tạo sàng nguyên tố của các số trong giới hạn
* Sau đó duyệt tham lam tìm cặp giá trị mà là số nguyên tố và cũng là số nguyên tố.

# ƯỚC SỐ - BỘI SỐ

## Lý thuyết chia hết

Định nghĩa 1: cho là các số nguyên. Ta nói chia hết (hay chia hết cho ) nếu tồn tại số nguyên sao cho .

* Ký hiệu:
* : đọc là chia hết .
* : đọc là chia hết cho .
* Khi (đọc là chia hết ), ta nói là ước của .
* Khi (đọc là chia hết cho ), ta nói là ước của .

Định nghĩa 2: ước chung lớn nhất của hai số và không đồng thời bằng là số nguyên dương lớn nhất chia hết cả và .

* Ký hiệu: đọc là ước chung lớn nhất của và .

Định nghĩa 3: bội chung nhỏ nhất của hai số nguyên và là số nguyên dương nhỏ nhất chia hết cho cả và .

* Ký hiệu:
* : đọc là bội chung nhỏ nhất của và .
* : đọc là bội chung nhỏ nhất của và .
* : đọc là bội chung nhỏ nhất của và .

Định nghĩa 4: các số nguyên và được gọi là nguyên tố cùng nhau nếu .

## Lý thuyết đồng dư

### Định nghĩa

Định nghĩa 5: cho và là các số nguyên, là số nguyên dương. Ta nói đồng dư mô đu lô nếu .

* Ký hiệu: đọc là đồng dư mô đu lô .
* Ví dụ 01: trong mô đun 12, ta có thể viết.
* Ví dụ 02: trong mô đun 5, ta có thể viết.

### Một số tính chất đồng dư

* Phản xạ: .
* Đối xứng: khi và chỉ khi với mọi .
* Bắc cầu: nếu và thì .
* Nếu và , hoặc thì:
* với mọi số nguyên .
* với mọi số nguyên khác .
* với mọi số nguyên .
* (bảo toàn phép cộng).
* (bảo toàn phép trừ).
* (bảo toàn phép nhân).
* với mọi số nguyên không âm (bảo toàn phép mũ).
* , với mọi đa thức có hệ số nguyên (bảo toàn với đa thức).
* Nếu , với là số nguyên bất kì, thì
* Nếu và nguyên tố cùng nhau với , thì .
* Nếu , thì .

Định lý fecmat nhỏ: cho nguyên tố và với không chia hết cho thì sẽ chia hết cho . Nghĩa là, .

* Ví dụ:
* .

## Số lượng ước số của một số nguyên dương

Bài toán: đếm số lượng ước số của một số nguyên dương.

* Ví dụ :
* .
* Các ước số của là: .
* Số lượng ước số của là: .
* Ví dụ :
* .
* Các ước số của là: .
* Số lượng ước số của là: .

Khai báo hàm đếm số lượng ước số của một số nguyên dương.

1. int countDivisors(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm countDivisors ta có giá trị của tham số thứ nhất chưa?

### Giải thuật ngây thơ đếm số lượng ước số

Ý tưởng giải thuật ngây thơ đếm số lượng ước số là duyệt qua các số nguyên từ tới và đếm số lượng ước số của .

* Định nghĩa hàm:

1. long long countDivisors\_1(long long n)
2. {
3. long long count = 0;
4. for (long long i = 1; i <= n; i++)
5. if (n % i == 0)
6. count++;
7. return count;
8. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

### Giải thuật đếm số lượng ước số dựa vào thuật toán phân tích thừa số nguyên tố

Phân tích ra thừa số nguyên tố.

|  |  |
| --- | --- |
| Giá trị | Thừa số nguyên tố |
| 1.000 | 2 |
| 500 | 2 |
| 250 | 2 |
| 125 | 5 |
| 25 | 5 |
| 5 | 5 |
| 1 |  |

Kết quả phân tích ra thừa số nguyên tố .

* Các ước số của là:
* Công thức đếm đếm số lượng ước số của số nguyên dương :
* Chứng minh
* Giả sử ta phân tích được thành các thừa số nguyên tố ở dạng:
* Các ước số của sẽ có dạng:
* Trong đó:
* Theo nguyên lý nhân, ta thấy:
  + có  cách chọn.
  + có cách chọn.
  + có cách chọn.
  + có cách chọn.
  + có cách chọn.
* Suy ra: .
* Kết luận: công thức đếm đếm số lượng ước số của số nguyên dương :
* Hãy đếm số lượng ước số của .
* .
* Phân tích ra thừa số nguyên tố.

|  |  |
| --- | --- |
| Giá trị | Thừa số nguyên tố |
| 1.000 | 2 |
| 500 | 2 |
| 250 | 2 |
| 125 | 5 |
| 25 | 5 |
| 5 | 5 |
| 1 |  |

* Do đó: .
* Số lượng ước số của là: .

Ví dụ: hãy đếm số lượng ước số của .

* .
* Phân tích ra thừa số nguyên tố.

|  |  |
| --- | --- |
| Giá trị | Thừa số nguyên tố |
| 24 | 2 |
| 12 | 2 |
| 6 | 2 |
| 3 | 3 |
| 1 |  |

* Do đó: .
* Số lượng ước số của là: .
* Từ đây ta có ý tưởng đếm số ước của một số nguyên dương dựa vào thuật toán phân tích thành thừa số nguyên tố như sau:
* Đặt một biến đếm để đếm là số lần xuất hiện của thừa số nguyên tố trong phân tích thừa số nguyên tố của .
* Nhân dồn các biến đếm của từng thừa số nguyên tố.
* Công thức đếm đếm số lượng ước số của số nguyên dương :
* Định nghĩa hàm:

1. long long countDivisors\_2(long long n)
2. {
3. long long T = 1;
4. long long p = 2;
5. while (n > 1)
6. {
7. long long count = 0;
8. while (n % p == 0)
9. {
10. count++;
11. n /= p;
12. }
13. T = T \* (count + 1);
14. p++;
15. }
16. return T;
17. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến trong tình huống là số nguyên tố.

* Viết với ngôn ngữ .

### Giải thuật đếm số lượng ước số dựa vào thuật toán phân tích thừa số nguyên tố trong đoạn

Xuất phát từ nhận xét sau: không thể xảy ra trường hợp mọi thừa số nguyên tố của đều lớn hơn , do đó chúng ta chỉ cần xét các ước số của từ tới và chia dần cho các ước của nó. Nếu không thể tìm được ước nào trong đoạn thì phải là một số nguyên tố.

* Công thức đếm đếm số lượng ước số của số nguyên dương :
* Định nghĩa hàm:

1. long long countDivisors\_3(long long n)
2. {
3. long long T = 1;
4. long long p = 2;
5. while (n > 1 && p \* p <= n)
6. {
7. long long count = 0;
8. while (n % p == 0)
9. {
10. count++;
11. n /= p;
12. }
13. T = T \* (count + 1);
14. p++;
15. }
16. if (n > 1)
17. T = T \* 2;
18. return T;
19. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến trong tình huống là số nguyên tố.

* Viết với ngôn ngữ .

### Giải thuật dựa vào phân tích thừa số nguyên tố với sàng Eratosthenes

* Giải thuật đếm số lượng ước số với sàng :
* Sử dụng thuật toán sàng tìm ước nguyên tố nhỏ nhất của mọi số nguyên trong đoạn .
* Chia cho ước nguyên tố nhỏ nhất của cho tới khi . Mỗi bước lặp đếm số lượng ước số liên quan số nguyên tố .
* Định nghĩa hàm:

1. long long countDivisors\_4(long long n)
2. {
3. vector<long long> minprime(n + 1);
4. for (long long p = 0; p <= n; p++)
5. minprime[p] = p;
6. for (long long p = 2; p \* p <= n; p++)
7. if (minprime[p] == p)
8. for (int i = p \* p; i <= n; i += p)
9. if (minprime[i] != i)
10. minprime[i] = p;
11. long long T = 1;
12. while (n > 1)
13. {
14. long long minp = minprime[n];
15. long long count = 0;
16. while (n % minp == 0)
17. {
18. count++;
19. n /= minp;
20. }
21. T = T \* (count + 1);
22. }
23. return T;
24. }

Giải thuật có độ phức tạp: do ta phải xây dựng sàng . Tuy nhiên, thao tác đếm số lượng ước số của chỉ mất độ phức tạp .

* Viết với ngôn ngữ .

## Tổng các ước số của một số nguyên dương

Bài toán: tính tổng các ước số của một số nguyên dương.

* Ví dụ :
* Các ước số của là: .
* Tổng ước số của là: .
* Ví dụ :
* Các ước số của là: .
* Tổng các ước số của là: .
* Khai báo hàm:

1. long long sumDivisors(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm sumDivisors ta có giá trị của tham số thứ nhất chưa?

### Giải thuật thơ ngây tính tổng ước số

Ý tưởng giải thuật thơ ngây tính tổng ước số là duyệt qua các số nguyên từ tới và tính tổng ước số của .

* Định nghĩa hàm:

1. long long sumDivisors\_1(long long n)
2. {
3. long long s = 0;
4. for (long long i = 1; i <= n; i++)
5. if (n % i == 0)
6. s = s + i;
7. return s;
8. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

### Giải thuật tính tổng ước số dựa vào thuật toán phân tích thừa số nguyên tố

Phân tích ra thừa số nguyên tố.

|  |  |
| --- | --- |
| Giá trị | Thừa số nguyên tố |
| 1.000 | 2 |
| 500 | 2 |
| 250 | 2 |
| 125 | 5 |
| 25 | 5 |
| 5 | 5 |
| 1 |  |

Vậy: .

* Giả sử ta phân tích được thành các thừa số nguyên tố ở dạng:
* Tổng các ước nguyên dương của được tính theo công thức sau:
* Chứng minh công thức:
* Các ước số của sẽ có dạng:
* Trong đó:
* Tổng cái ước số của sẽ là:
* Công thức của dãy cấp số nhân như sau:
* Do đó:
* Kết luận: tổng các ước nguyên dương của được tính theo công thức: .

Ví dụ : tính tổng các ước số của .

* Phân tích thừa số nguyên tố: .
* Tổng ước số của là: .

Ví dụ : tính tổng các ước số của .

* Phân tích thừa số nguyên tố: .
* Tổng ước số của là: .

Nhắc lại: công thức tính tổng các ước số của một số nguyên dương :

* Định nghĩa hàm:

1. long long sumDivisors\_2(long long n)
2. {
3. long long T = 1;
4. long long p = 2;
5. while (n > 1)
6. {
7. long long count = 0;
8. long long power = 1;
9. while (n % p == 0)
10. {
11. power \*= p;
12. count++;
13. n /= p;
14. }
15. T = T \* (power \* p - 1) / (p - 1);
16. p++;
17. }
18. return T;
19. }

* Viết với ngôn ngữ .

### Giải thuật tính tổng ước số dựa vào thuật toán phân tích thừa số nguyên tố trong đoạn

Xuất phát từ nhận xét sau: không thể xảy ra trường hợp mọi thừa số nguyên tố của đều lớn hơn , do đó chúng ta chỉ cần xét các ước số của từ tới và chia dần cho các ước của nó. Nếu không thể tìm được ước nào từ tới thì phải là một số nguyên tố.

* Định nghĩa hàm

1. long long sumDivisors\_3(long long n)
2. {
3. long long T = 1;
4. long long p = 2;
5. while (n > 1 && p \* p <= n)
6. {
7. long long count = 0;
8. long long power = 1;
9. while (n % p == 0)
10. {
11. power \*= p;
12. count++;
13. n /= p;
14. }
15. if (count > 0)
16. T = T \* (power \* p - 1) / (p - 1);
17. p++;
18. }
19. if (n > 1)
20. T = T \* (n \* n - 1) / (n - 1);
21. return T;
22. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến trong tình huống là số nguyên tố.

* Viết với ngôn ngữ .

### Giải thuật tính tổng ước số dựa vào phân tích thừa số nguyên tố với sàng Eratosthenes

* Giải thuật tính tổng ước số với sàng :
* Sử dụng thuật toán sàng tìm ước nguyên tố nhỏ nhất của mọi số nguyên trong đoạn .
* Chia cho ước nguyên tố nhỏ nhất của cho tới khi . Mỗi bước lặp tính tổng các ước số liên quan nguyên tố .
* Định nghĩa hàm:

1. long long sumDivisors\_4(long long n)
2. {
3. vector<long long> minprime(n + 1);
4. for (long long p = 0; p <= n; p++)
5. minprime[p] = p;
6. for (long long p = 2; p \* p <= n; p++)
7. if (minprime[p] == p)
8. for (int i = p \* p; i <= n; i += p)
9. if (minprime[i] == i)
10. minprime[i] = p;
11. long long T = 1;
12. while (n > 1)
13. {
14. long long minp = minprime[n];
15. long long count = 0;
16. long long power = 1;
17. while (n % minp == 0)
18. {
19. count++;
20. power \*= minp;
21. n /= minp;
22. }
23. T = T \* (power \* minp - 1) / (minp - 1);
24. }
25. return T;
26. }

Giải thuật có độ phức tạp: do ta phải xây dựng sàng . Tuy nhiên, thao tác phân tích một số thành thừa số nguyên tố chỉ mất độ phức tạp .

* Viết với ngôn ngữ .

## Ước số chung lớn nhất của hai số nguyên dương

* Bài toán: tìm ước số chung lớn nhất của hai số nguyên dương và .

Khai báo hàm tìm ước số chung lớn nhất của hai số nguyên dương.

1. long long uscln(long long, long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ hai?
* Khi gọi hàm uscln ta có giá trị của tham số thứ nhất chưa?
* Khi gọi hàm uscln ta có giá trị của tham số thứ hai chưa?

### Phương pháp hiệu

* Viết với ngôn ngữ .

1. long long uscln\_1(long long a, long long b)
2. {
3. a = abs(a);
4. b = abs(b);
5. while (a != 0 && b != 0)
6. {
7. if (a > b)
8. a = a - b;
9. else
10. b = b - a;
11. }
12. return a + b;
13. }

Chạy từng bước với .

* Điều kiện lặp Đ
  + if S
  + else
* Điều kiện lặp Đ
  + if Đ
* Điều kiện lặp Đ
  + if S
  + else
* Điều kiện lặp S
* Xuất
* Viết với ngôn ngữ .

### Phương pháp chia lấy dư

1. long long uscln\_2(long long a, long long b)
2. {
3. a = abs(a);
4. b = abs(b);
5. while (a != 0 && b != 0)
6. {
7. if (a > b)
8. a = a % b;
9. else
10. b = b % a;
11. }
12. return a + b;
13. }

Chạy từng bước với .

* Điều kiện lặp Đ
  + if S
  + else
* Điều kiện lặp Đ
  + if Đ
* Điều kiện lặp S
* Xuất

### Phương pháp Euclid

Bài toán: tìm ước số chung lớn nhất của hai số nguyên không âm và .

* Thuật toán Euclid
* Đặt: , ,
* Ta thực hiện phép chia:

, ,

* Dừng lại khi , nếu ta tiếp tục:

, ,

* Dừng lại khi , nếu ta tiếp tục:

, ,

* Dừng lại khi , nếu ta tiếp tục
* Dừng lại khi , nếu ta tiếp tục

, ,

* Dừng lại khi

, ,

* Khi đó:

Ước số chung lớn nhất () của 2 số nguyên dương được tính theo thuật toán Euclid:

* Định nghĩa hàm:

1. long long uscln\_3(long long a, long long b)
2. {
3. a = abs(a);
4. b = abs(b);
5. while (a != 0 && b != 0)
6. {
7. a = a % b;
8. swap(a, b);
9. }
10. return a + b;
11. }

* Viết với ngôn ngữ .

Chạy từng bước với .

* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp S
* Xuất

Cái tiến hàm trên một xíu:

1. long long uscln(long long a, long long b)
2. {
3. a = abs(a);
4. b = abs(b);
5. while (b > 0)
6. {
7. a = a % b;
8. swap(a, b);
9. }
10. return a;
11. }

* Viết với ngôn ngữ .

Chạy từng bước với .

* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp Đ
* Điều kiện lặp S
* Xuất

## Bội số chung nhỏ nhất của hai số nguyên dương

Bội số chung nhỏ nhất () của 2 số nguyên dương được tính theo công thức:

Khai báo hàm tìm bội số chung nhỏ nhất của hai số nguyên dương:

1. long long bscnn(long long, long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ hai?
* Khi gọi hàm bscnn ta có giá trị của tham số thứ nhất chưa?
* Khi gọi hàm bscnn ta có giá trị của tham số thứ hai chưa?

Định nghĩa hàm tìm bội số chung nhỏ nhất của hai số nguyên dương:

1. long long bscnn\_1(long long a, long long b)
2. {
3. return (a \* b) / uscln(a, b);
4. }

* Viết với ngôn ngữ .
* Cái tiến hàm trên một xíu:

1. long long bscnn\_2(long long a, long long b)
2. {
3. return a / uscln(a, b) \* b;
4. }

* Viết với ngôn ngữ .

## Hàm phi Euler

Phi hàm Euler: xác định số lượng các số nguyên dương nhỏ hơn và nguyên tố cùng nhau với .

* Ví dụ:
* Giải thích: vì có sáu số 1, 2, 4, 5, 7 và 8 là nguyên tố cùng nhau với 9.

## Bài tập ước số, bội số

1. Tính tổng các ước dương của .
2. Hai số tự nhiên được coi là hữu nghị nếu như số này bằng tổng các ước số của số kia và ngược lại. Định nghĩa hàm tìm các cặp số hữu nghị trong đoạn .

* Số hoàn thiện luôn luôn là số hữu nghị của chính nó.

1. Đếm số lượng ước số của .

* Ví dụ:
* Dữ liệu vào: .
* Dữ liệu ra: .
* Giải thích: .

1. Cho số nguyên dương , hãy tìm số nguyên dương nhỏ nhất sao cho chia hết cho .
2. Đếm số lượng chữ số tận cùng của
3. Tìm chữ số khác cuối cùng của
4. Tìm hai chữ số khác cuối cùng của

# LÝ THUYẾT TẬP HỢP

## Các phép toán trên tập hợp

* Phần bù của trong , kí hiệu , là tập hợp các phần tử của không thuộc :
* Hợp của và , kí hiệu , là tập hợp các phần tử hoặc thuộc vào hoặc thuộc vào :
* Giao của và , kí hiệu , là tập hợp các phần tử đồng thời thuộc cả và :
* Hiệu của và , kí hiệu , là tập hợp các phần tử thuộc tập nhưng không thuộc :

## Các tính chất của phép toán trên tập hợp

* Kết hợp:
* Giao hoán:
* Phân bố:
* Đối ngẫu:

## Tích Đề-các của các tập hợp

* Tích đề-các ghép hai tập hợp:
* Tích Đề-các mở rộng ghép nhiều tập hợp:

## Nguyên lí cộng

* Nếu và là hai tập hợp rời nhau thì
* Nguyên lí cộng mở rộng cho nhiều tập hợp đôi một rời nhau:
* Nếu là một phân hoạch của tập thì:

## Nguyên bù trừ

* Nếu và không rời nhau thì
* Nguyên lí mở rộng cho nhiều tập hợp:
* Giả sử là các tập hữu hạn:
* Trong đó là tổng phần tử của tất cả các giao của tập lấy từ tập đã cho.

## Nguyên lí nhân

* Nếu mỗi thành phần của bộ có thứ tự thành phần có khả năng lựa chọn , thì số bộ sẽ được tạo ra là tích số của các khả năng này .
* Một hệ quả trực tiếp của nguyên lí nhân:

## Chỉnh hợp lặp

* Xét tập hữu hạn gồm phần tử .
* Một chỉnh hợp lặp chập của phần tử là một bộ có thứ tự gồm phần tử của , các phần tử có thể lặp lại.
* Một chỉnh hợp lặp chập của có thể xem như một phần tử của tích Đềcac .
* Theo nguyên lí nhân, số tất cả các chỉnh hợp lặp chập của sẽ là .

## Chỉnh hợp không lặp

* Một chỉnh hợp không lặp chập của phần tử là một bộ có thứ tự gồm thành phần lấy từ phần tử của tập đã cho. Các thành phần không ñược lặp lại. Để xây dựng một chỉnh hợp không lặp, ta xây dựng dần từng thành phần đầu tiên. Thành phần này có khả năng lựa chọn. Mỗi thành phần tiếp theo, số khả năng lựa chọn giảm đi 1 so với thành phần đứng trước, do đó, theo nguyên lí nhân, số chỉnh hợp không lặp chập của sẽ là

## Hoán vị

* Một hoán vị của phần tử là một cách xếp thứ tự các phần tử đó. Một hoán vị của phần tử được xem như một trường hợp riêng của chỉnh hợp không lặp khi . Do đó số hoán vị của phần tử là !

## Tổ hợp

* Một tổ hợp chập của phần tử là một bộ không kể thứ tự gồm thành phần khác nhau lấy từ phần tử của tập đã cho.
* Một số tính chất:
* . Tổ hợp chập của phần tử bằng tổ hợp chập của phần tử.
* . Tổ hợp chập của phần tử bằng tổ hợp chập của phần tử và bằng .
* . Tổ hợp chập của phần tử bằng tổ hợp chập của phần tử cộng với tổ hợp chập của .

1. Viết chương trình tính tổ hợp chập của phần tử

* Giới hạn:
* Dữ liệu vào:
  + Lần lượt là hai số cách nhau bởi khoảng trẳng
* Dữ liệu ra:
  + Kết quả tổ hợp chập của .

Chương trình.

1. #include <iostream>
2. using namespace std;
3. long ToHop(int n, int k)
4. {
5. if (k > n - k)
6. k = n - k; // C(n, k) = C(n, n-k)
7. long temp = 1;
8. for (int i = 0; i < k; i++)
9. {
10. temp \*= (n - i);
11. temp /= (i + 1);
12. }
13. return temp;
14. }
15. int main()
16. {
17. int n;
18. cin >> n;
19. int k;
20. cin >> k;
21. cout << ToHop(n, k);
22. return 1;
23. }

* Viết với ngôn ngữ .

## Bài tập lý thuyết tập hợp

# CÁC SỐ ĐẶC BIỆT

## Số fibonacci

* Số được xác định bởi công thức sau:
* Một số phần tử đầu tiên của dãy số :

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  | … |
|  |  |  |  |  |  |  |  |  |  |  |  | … |

Khai báo hàm tính số hạng thứ của dãy :

1. long long Fibonacci(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm Fibonacci ta có giá trị của tham số thứ nhất chưa?

### Phương pháp sử dụng bộ nhớ phụ tính số hạng thứ n của dãy fibonacci

* Phương pháp sử dụng bộ nhớ phụ để lưu các giá trị của dãy .

Định nghĩa hàm tính số hạng thứ của dãy :

1. long long Fibonacci(long long n)
2. {
3. vector<long long> fibo(n + 1);
4. fibo[0] = 0;
5. fibo[1] = 1;
6. for (long long i = 2; i <= n; i++)
7. fibo[i] = fibo[i - 1] + fibo[i - 2];
8. return fibo[n];
9. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Độ phức tạp bộ nhớ của hàm cài đặt trên là vì ta cần một bộ nhớ phụ có kích thước để lưu giá trị .
* Viết với ngôn ngữ .

### Phương pháp không sử dụng bộ nhớ phụ tính số hạng thứ n của dãy fibonacci

Định nghĩa hàm tính số hạng thứ của dãy không sử dụng bộ nhớ phụ.

1. long long Fibonacci(long long n)
2. {
4. if (n == 0)
5. return 0;
6. if (n == 1)
7. return 1;
8. long long Ftt = 0;
9. long long Ft = 1;
10. long long i = 2;
11. long long Fhh = 0;
12. while (i <= n)
13. {
14. Fhh = Ft + Ftt;
15. i++;
16. Ftt = Ft;
17. Ft = Fhh;
18. }
19. return Fhh;
20. }

Hàm cài đặt có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Độ phức tạp bộ nhớ của hàm cài đặt trên là .
* Viết với ngôn ngữ .

### Phương pháp đệ quy tính số hạng thứ n của dãy fibonacci

Định nghĩa hàm tính số hạng thứ của dãy bằng phương pháp đệ quy.

1. long long Fibonacci(long long n)
2. {
3. if (n == 0)
4. return 0;
5. if (n == 1)
6. return 1;
7. return Fibonacci(n - 1) + Fibonacci(n - 2);
8. }

* Viết với ngôn ngữ .

### Các tính chất của số fibonacci

* .
* có chữ số tận cùng bằng khi và chỉ khi .
* có chữ số tận cùng hai chữ số khi và chỉ khi .
* là số nguyên tố thì cũng là số nguyên tố .

## Số Catalan

* Số được xác định bởi công thức sau:
* Một số phần tử đầu tiên của dãy số :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Khai báo hàm tính số hạng thứ của dãy :

1. long long Catalan(long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm Catalan ta có giá trị của tham số thứ nhất chưa?
* Dãy số :

### Tính số hạng thứ n của dãy Catalan theo định nghĩa

* Số được xác định bởi công thức sau:

Định nghĩa hàm tính số hạng thứ của dãy theo định nghĩa:

1. long long GiaiThua(long long n)
2. {
3. long long T = 1;
4. for (long long i = 1; i <= n; i++)
5. T = T \* i;
6. return T;
7. }
8. long long Catalan(long long n)
9. {
10. long long a = GiaiThua(2 \* n);
11. long long b = GiaiThua(n);
12. long long c = GiaiThua(n + 1);
13. return a / (b \* c);
14. }

* Cải tiến hàm Catalan lần 1.

1. long long Catalan(long long n)
2. {
3. long long a = GiaiThua(2 \* n);
4. long long b = GiaiThua(n);
5. long long c = GiaiThua(n + 1);
6. return a / b / c;
7. }

* Cải tiến hàm Catalan lần 2.

1. long long Catalan(long long n)
2. {
3. long long a = GiaiThua(2 \* n);
4. long long b = GiaiThua(n);
5. return a / b / b / (n + 1);
6. }

* Viết với ngôn ngữ .

### Tính số hạng thứ n của dãy Catalan bằng phương pháp sử dụng bộ nhớ phụ

* Công thức truy hồi tính số hạng thứ của dãy :
* Phương pháp sử dụng bộ nhớ phụ để lưu các giá trị của dãy và sử dụng công thức truy hồi

Định nghĩa hàm tính số hạng thứ của dãy bằng phương pháp sử dụng bộ nhớ phụ:

1. long long Catalan(long long n)
2. {
3. vector<long long> Cata(n + 1);
4. Cata[0] = Cata[1] = 1;
5. for (long long k = 2; k <= n; k++)
6. {
7. Cata[k] = 0;
8. for (int i = 0; i < k; i++)
9. Cata[k] += Cata[i] \* Cata[k-i-1];
10. }
11. return Cata[n];
12. }

* Viết với ngôn ngữ .

### Tính số hạng thứ n của dãy Catalan bằng phương pháp không sử dụng bộ nhớ phụ

* Công thức:
* Khai triển công thức:

Định nghĩa hàm tính số hạng thứ của dãy bằng phương pháp không sử dụng bộ nhớ phụ:

1. long long binomialCoeff(long long n,long long k)
2. {
3. if (k > n - k)
4. k = n - k; // C(n,k) = C(n,n-k)
5. long long temp = 1;
6. for (long long i = 0; i < k; ++i)
7. {
8. temp \*= (n - i);
9. temp /= (i + 1);
10. }
11. return temp;
12. }
13. long long Catalan(long long n)
14. {
15. return binomialCoeff(2 \* n, n) / (n + 1);
16. }

* Viết với ngôn ngữ .

### Tính số hạng thứ n của dãy Catalan bằng phương pháp đệ quy

* Công thức truy hồi tính số hạng thứ của dãy :

Định nghĩa hàm tính số hạng thứ n của dãy Catalan bằng phương pháp đệ quy:

1. long long Catalan(long long n)
2. {
3. if (n <= 1)
4. return 1;
5. long long temp = 0;
6. for (int i = 0; i < n; i++)
7. temp += Catalan(i) \* Catalan(n - i - 1);
8. return temp;
9. }

## Số chính phương

## Số phong phú

## Bài tập các số đặc biệt

1. Đếm số lượng số chính phương không vượt quá .
2. Kiểm tra có phải là số đối xứng không?
3. Tìm chữ số lớn nhất của .

# MỘT SỐ THUẬT TOÁN

## Phép nhân Ai cập

Bài toán: tính với .

* Ý tưởng giải thuật nhân thực hiện như sau:

### Giải thuật nhân Ai cập

* Phép nhân là một phương pháp để nhân hai số không cần bảng cửu chương, chỉ sử dụng phép toán nhân đôi, phép toán chia đôi và phép toán cộng.

Phép nhân nhân hai số hạng bằng cách phân tách một trong hai số hạng (tốt nhất là số hạng nhỏ nhất) thành tổng các giá trị lũy thừa của , sau đó tạo bảng nhân đôi của số hạng thứ hai, kết quả của phép nhân hai số hạng là phép cộng các giá trị một cách hợp lý của bảng nhân đôi.

* Phép nhân còn được gọi là phép nhân Ethiopia, phép nhân Nga hoặc phép nhân Nông dân.

Tính

* Bảng lũy thừa .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Lũy thừa | Giá trị | Phân tách | Bội | Tính |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* Phân tách:
* Tính:

Khai báo hàm:

1. long long mulEgypt(long long, long long);

### Hàm cài đặt xử lý trên số nguyên

* Ý tưởng giải thuật nhân :

Định nghĩa hàm cài đặt thuật toán nhân Ai cập xử lý trên số nguyên:

1. long long mulEgypt(long long a, long long b)
2. {
3. long long t = 0;
4. while (b >= 1)
5. {
6. if (b % 2 != 0)
7. t = t + a;
8. b = b / 2;
9. a = a \* 2;
10. }
11. return t;
12. }

Giải thuật có độ phức tạp: do ở mỗi bước lặp ta chia cho 2.

* Viết với ngôn ngữ .
* Chạy từng bước với .
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp Đ
  + If S
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp S
* Xuất
* Chạy từng bước với bằng cách lập bảng.

|  |  |  |
| --- | --- | --- |
| 238 | **13** | 0 |
| 476 | **6** | 238 |
| 952 | **3** |  |
| 1.904 | **1** | 952 + 238 |
| 3.808 | **0** | 1.904 + 952 + 238 |

### Hàm cài đặt xử lý trên bit

Định nghĩa hàm cài đặt thuật toán nhân xử lý trên bit:

1. long long mulEgypt(long long a, long long b)
2. {
3. long long t = 0;
4. while (b >= 1)
5. {
6. if (b & 1)
7. t = t + a;
8. b >>= 1;
9. a <<= 1;
10. }
11. return t;
12. }

Hàm cài đặt vẫn có độ phức tạp: do ở mỗi bước lặp ta chia cho . Tuy nhiên, hàm thực hiện nhanh hơn nhờ chuyển đổi các phép toán nhân đôi, chia đôi thành phép dịch phải bit, phép dịch trái bit. Hơn nữa việc kiểm tra số nguyên là số lẻ dựa vào phép toán trên bit.

* Viết với ngôn ngữ .

### Hàm cài đặt đệ quy

* Ý tưởng giải thuật nhân thực hiện như sau:

Định nghĩa hàm cài đặt thuật toán nhân bằng kỹ thuật đệ quy:

1. long long mulEgypt(long long a, long long b)
2. {
3. if (b == 0)
4. return 0;
5. long long t = mulEgypt(a, b / 2);
6. if (b % 2 != 0)
7. return t + t + a;
8. return t + t;
9. }

Giải thuật có độ phức tạp: do ở mỗi lần gọi đệ quy ta chia cho 2.

* Viết với ngôn ngữ .

## Tính với

Bài toán: tính với .

* Nhận xét: nếu thực hiện phép toán nhân trước thì sẽ bị tràn số.
* Ví dụ: tính .
* Vậy:

### Hàm cài đặt xử lý trên số nguyên

Định nghĩa hàm:

1. long long mulEgyptModulo(long long a,
2. long long b, long long M)
3. {
4. long long t = 0;
5. while (b >= 1)
6. {
7. if (b % 2 != 0)
8. t = (t % M + a % M) % M;
9. b = b / 2;
10. a = a \* 2;
11. }
12. return t;
13. }

Giải thuật có độ phức tạp: do ở mỗi bước lặp ta chia cho 2.

* Viết với ngôn ngữ .

### Hàm cài đặt đệ quy

Định nghĩa hàm tính với .

1. long long mulEgyptModulo(long long a,
2. long long b, long long M)
3. {
4. if (b == 0)
5. return 0;
6. long long t=mulEgyptModulo(a,b/2,M) % M;
7. if (b % 2 != 0)
8. return (t + t + a % M) % M;
9. return (t + t) % M;
10. }

Giải thuật có độ phức tạp: do ở mỗi lần gọi đệ quy ta chia cho .

* Viết với ngôn ngữ .

## Tính lũy thừa , tính mũ

Bài toán 01: tính .

* Ví dụ :
* Dữ liệu vào:
  + .
* Dữ liệu ra:

Bài toán 02: tính .

* Ví dụ :
* Dữ liệu vào:
  + .
* Dữ liệu ra:

Khai báo hàm tính lũy thừa.

1. float LuyThua(float, long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ hai?
* Khi gọi hàm LuyThua ta có giá trị của tham số thứ nhất chưa?
* Khi gọi hàm LuyThua ta có giá trị của tham số thứ hai chưa?

Khai báo hàm tính mũ.

1. long long power(long long, long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Khi gọi hàm power ta có giá trị của tham số thứ nhất chưa?
* Khi gọi hàm power ta có giá trị của tham số thứ hai chưa?

### Giải thuật tự nhiên tính lũy thừa, tính mũ

Định nghĩa hàm tính lũy thừa.

1. float LuyThua(float x, long n)
2. {
3. float t = 1;
4. for (int i = 1; i <= n; i++)
5. t = t \* x;
6. return t;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Trường hợp máy tính cá nhân (laptop) tại thời điểm năm 2023 chạy mất khoảng 4s. Điều này còn phụ thuộc vào cấu hình máy, các bạn tự viết chương trình test điều này.
* Viết với ngôn ngữ .

Định nghĩa hàm tính mũ.

1. long long power(long long a, long long b)
2. {
3. long long t = 1;
4. for (long long i = 1; i <= b; i++)
5. t = t \* a;
6. return t;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

### Phương pháp đệ quy tính lũy thừa, tính mũ

Định nghĩa hàm đệ quy tính lũy thừa.

1. float LuyThua(float x, long n)
2. {
3. if (n == 0)
4. return 1;
5. return x \* LuyThua(x, n - 1);
6. }

Giải thuật có độ phức tạp: do mỗi bước gọi đệ quy ta giảm xuống .

* Trường hợp hàm đệ quy không thực hiện được trên máy tính cá nhân thông thường.
* Viết với ngôn ngữ .

Định nghĩa hàm đệ quy tính mũ.

1. long long power(long long a, long long b)
2. {
3. if (b == 0)
4. return 1;
5. return a \* power(a, b - 1);
6. }

Giải thuật có độ phức tạp: do mỗi bước gọi đệ quy ta giảm xuống .

* Viết với ngôn ngữ .

### Phương pháp tính lũy thừa nhanh, tính mũ nhanh

Ý tưởng phương pháp tính lũy thừa nhanh:

Định nghĩa hàm tính lũy thừa nhanh.

1. float LuyThua(float x, long n)
2. {
3. float t = 1;
4. while (n > 0)
5. {
6. if (n % 2 != 0)
7. t = t \* x;
8. x = x \* x;
9. n = n / 2;
10. }
11. return t;
12. }

Giải thuật có độ phức tạp: do ở mỗi bước lặp ta chia cho 2.

* Viết với ngôn ngữ .
* Chạy từng bước với .
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp Đ
  + If S
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp Đ
  + If S
* Điều kiện lặp Đ
  + If S
* Điều kiện lặp Đ
  + If Đ
* Điều kiện lặp S
* Xuất

Định nghĩa hàm tính mũ nhanh.

1. long long power(long long a, long long b)
2. {
3. long long t = 1LL;
4. while (b > 0)
5. {
6. if (b % 2 != 0)
7. t = t \* a;
8. b = b / 2;
9. a = a \* a;
10. }
11. return t;
12. }

Giải thuật có độ phức tạp: do ở mỗi bước lặp ta chia cho 2.

* Viết với ngôn ngữ .

### Phương pháp tính lũy thừa nhanh, , tính mũ nhanh bằng đệ quy

Ý tưởng phương pháp tính lũy thừa nhanh:

Định nghĩa hàm phương pháp tính lũy thừa nhanh bằng đệ quy

1. float LuyThua(float x, long n)
2. {
3. if (n == 0)
4. return 1;
5. float t = LuyThua(x, n / 2);
6. if (n % 2 == 0)
7. return t \* t;
8. return t \* t \* x;
9. }

Giải thuật có độ phức tạp: do ở mỗi lần gọi đệ quy ta chia cho .

* Trường hợp tính lũy thừa nhanh bằng đệ quy thực hiện được trên máy tính cá nhân thông thường.
* Viết với ngôn ngữ .

Định nghĩa hàm phương pháp tính mũ bằng đệ quy

1. long long power(long long a, long long b)
2. {
3. if (b == 0)
4. return 1LL;
5. long long t = power(a, b / 2);
6. if (b % 2 == 0)
7. return t \* t;
8. return t \* t \* a;
9. }

Giải thuật có độ phức tạp: do ở mỗi lần gọi đệ quy ta chia cho .

* Viết với ngôn ngữ .

## Tính

Bài toán 01: tính .

* Ví dụ :
* Dữ liệu vào:
  + .
  + .
* Dữ liệu ra:

Khai báo hàm.

1. long long powerModulo(long long, long long,
2. long long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ hai?
* Tham số thứ ba kiểu dữ liệu là gì?
* Loại tham số thứ ba?
* Khi gọi hàm powerModulo ta có giá trị của tham số thứ nhất, thứ hai, thứ ba chưa?

### Giải thuật tự nhiên

* Giải thuật tự nhiên là ta đi tính xong rồi thực hiện phép toán với kết quả tìm được.

Định nghĩa hàm

1. long long powerModulo(long long a,
2. long long b, long long M)
3. {
4. long long t = 1;
5. while (b > 0)
6. {
7. if (b % 2 == 1)
8. t = t \* a;
9. b = b / 2;
10. a = a \* a;
11. }
12. return t % M;
13. }

Giải thuật có độ phức tạp: do ở mỗi lần lặp ta chia cho .

* Viết với ngôn ngữ .

### Giải thuật cải tiến

* Nhận xét: nếu thực hiện phép lũy thừa xong rồi thực hiện phép toán thì sẽ bị tràn số.

Định nghĩa hàm.

1. long long powerModulo(long long a,
2. long long b, long long M)
3. {
4. long long t = 1;
5. a = a % M;
6. while (b > 0)
7. {
8. if (b % 2 == 1)
9. t = (t \* a) % M;
10. b = b / 2;
11. a = (a \* a) % M;
12. }
13. return t;
14. }

Giải thuật có độ phức tạp: do ở mỗi lần lặp ta chia cho .

* Viết với ngôn ngữ .

### Giải thuật cải tiến đệ quy

Định nghĩa hàm

1. long long powerModulo(long long a,
2. long long b, long long M)
3. {
4. if (b == 0)
5. return 1LL;
6. long long t = powerModulo(a,b/2LL,M) % M;
7. t = mulEgyptModulo(t, t, M);
8. if (b % 2 == 0)
9. return t;
10. return mulEgyptModulo(t, a, M);
11. }

Giải thuật có độ phức tạp: do ở mỗi lần gọi đệ quy ta chia cho .

* Viết với ngôn ngữ .

## Một số đẳng thức cần biết

### Tổng từ 1 tới n

Bài toán:

* Ví dụ: tính .
* .
* .

Khai báo hàm.

1. long long Tong(long long);

#### Phương pháp truyền thống

* Sử dụng một biến chạy chạy từ tới . Mỗi bước lặp cộng dồn vào trong biến .

Định nghĩa hàm.

1. long long Tong(long long n)
2. {
3. long long s = 0;
4. for (long long i = 1; i <= n; i++)
5. s = s + i;
6. return s;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

#### Sử dụng công thức toán học

* Tổng được tính theo công thức sau:

Định nghĩa hàm.

1. long long Tong(long long n)
2. {
3. if (n % 2 == 0)
4. return n / 2 \* (n + 1);
5. return (n + 1) / 2 \* n;
6. }

Giải thuật có độ phức tạp: .

* Viết với ngôn ngữ .

### Tổng các bình phương

Bài toán:

* Ví dụ: tính .
* .
* .

Khai báo hàm.

1. long long TongBinhPhuong(long long);

#### Phương pháp truyền thống

* Sử dụng một biến chạy chạy từ tới . Mỗi bước lặp cộng dồn vào trong biến .

Định nghĩa hàm.

1. long long TongBinhPhuong(long long n)
2. {
3. long long s = 0;
4. for (long long i = 1; i <= n; i++)
5. s = s + i \* i;
6. return s;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

#### Sử dụng công thức toán học

* Tổng được tính theo công thức sau:

Định nghĩa hàm.

1. long long TongBinhPhuong(long long n)
2. {
3. return n \* (n + 1) \* (2 \* n + 1) / 6;
4. }

Giải thuật có độ phức tạp: .

* Viết với ngôn ngữ .

### Tổng các lập phương

Bài toán:

* Ví dụ: tính .
* .
* .

Khai báo hàm.

1. long long TongLapPhuong(long long);

#### Phương pháp truyền thống

* Sử dụng một biến chạy chạy từ tới . Mỗi bước lặp cộng dồn vào trong biến .

Định nghĩa hàm.

1. long long TongLapPhuong(long long n)
2. {
3. long long s = 0;
4. for (long long i = 1; i <= n; i++)
5. s = s + i \* i \* i;
6. return s;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

#### Sử dụng công thức toán học

* Tổng được tính theo công thức sau:

Định nghĩa hàm.

1. long long TongLapPhuong(long long n)
2. {
3. if (n % 2 == 0)
4. return (n/2)\*(n/2)\*(n+1)\*(n+1);
5. return ((n+1)/2)\*((n+1)/2)\*n\*n;
6. }

Giải thuật có độ phức tạp: .

* Viết với ngôn ngữ .

### Tổng các số lẻ

Bài toán:

* Ví dụ: tính .
* .
* .

Khai báo hàm.

1. long long TongLe(long long);

#### Phương pháp truyền thống

* Sử dụng một biến chạy chạy từ tới . Mỗi bước lặp cộng dồn vào trong biến .

Định nghĩa hàm.

1. long long TongLe(long long n)
2. {
3. long long s = 0;
4. for (long long i = 1; i<=2\*n+1; i += 2)
5. s = s + i;
6. return s;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

#### Sử dụng công thức toán học

* Tổng được tính theo công thức sau:

Định nghĩa hàm.

1. long long TongLe(long long n)
2. {
3. return (n + 1) \* (n + 1);
4. }

Giải thuật có độ phức tạp: .

* Viết với ngôn ngữ .

### Tổng các số chẵn

Bài toán:

* Ví dụ: tính .
* .
* .

Khai báo hàm.

1. long long TongChan(long long);

#### Phương pháp truyền thống

* Sử dụng một biến chạy chạy từ tới . Mỗi bước lặp cộng dồn vào trong biến .

Định nghĩa hàm.

1. long long TongChan(long long n)
2. {
3. long long s = 0;
4. for (long long i = 2; i <= 2 \* n; i += 2)
5. s = s + i;
6. return s;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

#### Sử dụng công thức toán học

* Tổng được tính theo công thức sau:

Định nghĩa hàm

1. long long TongChan(long long n)
2. {
3. return n \* (n + 1);
4. }

Giải thuật có độ phức tạp: .

* Viết với ngôn ngữ .

### Tổng dãy số có giá trị bắt đầu là 2 bước nhảy là 3

Bài toán:

* Ví dụ: tính .
* .
* .

Khai báo hàm.

1. long long TongBa(long long);

#### Phương pháp truyền thống

* Sử dụng một biến chạy chạy từ tới . Mỗi bước lặp cộng dồn vào trong biến .

Định nghĩa hàm.

1. long long TongBa(long long n)
2. {
3. long long s = 0;
4. for (long long i = 2; i <= 3\*n-1; i += 3)
5. s = s + i;
6. return s;
7. }

Giải thuật có độ phức tạp: do ta phải duyệt hết các số từ đến .

* Viết với ngôn ngữ .

#### Sử dụng công thức toán học

* Tổng được tính theo công thức sau:

Định nghĩa hàm.

1. long long TongBaNhanh(long long n)
2. {
3. if (n % 2 == 0)
4. return (n / 2) \* (3 \* n + 1);
5. return ((3 \* n + 1) / 2) \* n;
6. }

Giải thuật có độ phức tạp: .

* Viết với ngôn ngữ .

## Một số bất đẳng thức cần biết

### Bất đẳng thức AM – GM

### Bất đẳng thức Cauchy–Bunyakovsky–Schwarz

## Bài tập các thuật giải cơ sở

1. Cho hai số nguyên và với . Tìm chữ số tận cùng của .
2. Cho hai số nguyên và với . Tính .
3. Tính

* Ta có:
* Tổng quát:
* Hàm cài đặt:

1. long long Tong(int n)
2. {
3. long long s = 1;
4. for (int i = 1; i <= n; i++)
5. if (i % 2 != 0)
6. s = s - (3\*i-1) + (3\*i) - (3\*i+1);
7. else
8. s = s + (3\*i-1) - (3\*i) + (3\*i+1);
9. return s;
10. }

* Hàm cài đặt (Vicky):

1. long long Tong(int n)
2. {
3. long long s = 0;
4. for (int i = 1; i <= 3 \* n + 1; i++)
5. if (i % 2 != 0)
6. s = s + i;
7. else
8. s = s - i;
9. return s;
10. }

* Viết với ngôn ngữ .

# SỐ NGUYÊN LỚN

## Biểu diễn số nguyên lớn

Có 4 cách biểu diễn số nguyên lớn

* Chuỗi ký tự.

1. typedef string BigNum;

* Mảng các số.
* Véc tơ các số nguyên.
* Danh sách liên kết các số.

## Các hàm cơ bản xử lý số lớn

* Bảng chuyển đổi số thành ký tự:

| Số | Ký tự | Mã ascii |
| --- | --- | --- |
| 0 | '0' | 48 |
| 1 | '1' | 49 |
| 2 | '2' | 50 |
| 3 | '3' | 51 |
| 4 | '4' | 52 |
| 5 | '5' | 53 |
| 6 | '6' | 54 |
| 7 | '7' | 55 |
| 8 | '8' | 56 |
| 9 | '9' | 57 |
| 10 | 'A' | 65 |
| 11 | 'B' | 66 |
| 12 | 'C' | 67 |
| 13 | 'D' | 68 |
| 14 | 'E' | 69 |
| 15 | 'F' | 70 |

### Hàm đổi số thành ký tự

* Khai báo hàm:

1. char toChar(int);

* Định nghĩa hàm:

1. char toChar(int n)
2. {
3. char Letter[] = { '0','1','2','3','4','5',
4. '6','7','8','9','A','B','C','D','E','F' };
5. return Letter[n];
6. }

* Viết với ngôn ngữ .

### Hàm đổi ký tự thành số

* Khai báo hàm:

1. int toNumber(char);

* Định nghĩa hàm:

1. int toNumber(char c)
2. {
3. if (c == '0')
4. return 0;
5. if (c == '1')
6. return 1;
7. if (c == '2')
8. return 2;
9. if (c == '3')
10. return 3;
11. if (c == '4')
12. return 4;
13. if (c == '5')
14. return 5;
15. if (c == '6')
16. return 6;
17. if (c == '7')
18. return 7;
19. if (c == '8')
20. return 8;
21. if (c == '9')
22. return 9;
23. if (c == 'A' || c == 'a')
24. return 10;
25. if (c == 'B' || c == 'b')
26. return 11;
27. if (c == 'C' || c == 'c')
28. return 12;
29. if (c == 'D' || c == 'd')
30. return 13;
31. if (c == 'E' || c == 'e')
32. return 14;
33. return 15;
34. }

* Viết với ngôn ngữ .

## Phép so sánh hai số nguyên lớn

Hãy thực hiện phép so sánh hai số nguyên lớn sau:

* Số nguyên lớn thứ nhất:
  + "1.234.000.000.000.000.000.000.000.000".
  + 28 chữ số, 24 chữ số 0.
* Số nguyên lớn thứ hai:
  + "915.000.000.000.000.000.000.000.000.000".
  + 27 chữ số, 24 chữ số 0.
* Kết quả: "1.234.000.000.000.000.000.000.000.000" > "915.000.000.000.000.000.000.000.000.000".

Tuy nhiên, nếu nhìn dưới góc độ kiểu dữ liệu chuỗi (string) thì chuỗi "1.234.000.000.000.000.000.000.000.000" nhỏ hơn "915.000.000.000.000.000.000.000.000.000".

* Hàm so sánh hai chuỗi thực hiện như sau:
* So sánh ký tự đầu tiên của hai chuỗi.
  + Nếu ký tự đầu tiên của chuỗi thứ nhất lớn hơn ký tự đầu tiên của chuỗi thứ hai thì chuỗi thứ nhất lớn hơn chuỗi thứ hai.
  + Nếu ký tự đầu tiên của chuỗi thứ nhất nhỏ hơn ký tự đầu tiên của chuỗi thứ hai thì chuỗi thứ nhất nhỏ hơn chuỗi thứ hai.
* Ký tự đầu tiên của hai chuỗi bằng nhau thì so sánh tiếp tới ký tự thứ hai.
* …
* Các ví dụ so sánh chuỗi:
* Ví dụ 01: chuỗi "234" nhỏ hơn chuỗi "57"
* Ví dụ 02: chuỗi "1110" lớn hơn chuỗi "111"

Ý tưởng so sánh hai chuỗi số nguyên lớn: để so sánh hai số nguyên lớn được biểu diễn bằng chuỗi kí tự, ta sẽ đưa trở về bài toán so sánh hai chuỗi ký tự bằng cách như sau:

* Trước tiên, ta thêm các chữ số (ký tự 0) vào đầu số có số chữ số nhỏ hơn để hai số nguyên lớn có số lượng chữ số bằng nhau.
* Sau đó sử dụng trực tiếp phép toán so sánh trên chuỗi kí tự để so sánh hai số nguyên lớn.

Các bước thực hiện khi so sánh số nguyên lớn.

* Số nguyên lớn thứ 1:

|  |
| --- |
| "1.234.000.000.000.000.000.000.000" |

* Số nguyên lớn thứ 2:

|  |
| --- |
| "915.000.000.000.000.000.000.000" |

* Số nguyên lớn thứ nhất có chữ số, số nguyên lớn thứ hai có chữ số.
* Thêm 1 chữ số 0 vào đầu số nguyên lớn thứ 2:

|  |
| --- |
| "0.915.000.000.000.000.000.000.000" |

* Bài toán so sánh hai số nguyên lớn trở về bài toán so sánh hai chuỗi ký tự.

Giá trị trả về : hàm so sánh hai số nguyên lớn trả về 1 trong 3 giá trị:

* Giá trị : số nguyên lớn thứ nhất nhỏ hơn số nguyên lớn thứ hai.
* Giá trị : số nguyên lớn thứ nhất bằng số nguyên lớn thứ hai.
* Giá trị : số nguyên lớn thứ nhất lớn hơn số nguyên lớn thứ hai.

Khai báo hàm so sánh hai số nguyên lớn:

1. int compareBigNum(BigNum, BigNum);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm compareBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Định nghĩa hàm so sánh hai số nguyên lớn:

1. int compareBigNum(BigNum x, BigNum y)
2. {
3. while (x.size() < y.size()) x = '0' + x;
4. while (x.size() > y.size()) y = '0' + y;
5. if (x > y)
6. return 1;
7. if (x < y)
8. return -1;
9. return 0;
10. }

* Lưu ý: câu lệnh if (x > y) và câu lệnh if (x < y) là hai câu lệnh so sánh trên chuỗi ký tự.
* Viết với ngôn ngữ .

## Phép cộng hai số nguyên lớn

Phép cộng hai số nguyên lớn:

* Thực hiện từ phải qua trái
* Phần nhớ được mang sang trái.
* Hãy thực hiện hiện phép cộng hai số nguyên lớn sau:

|  |  |  |  |
| --- | --- | --- | --- |
| “99.646.234.875.495.327.686.473" |  |  |  |
| "927.234.572.289.123.934.865" |  |  |  |

* Thêm các ký tự 0 hợp lý để hai số nguyên lớn có độ dài bằng nhau.

|  |  |  |  |
| --- | --- | --- | --- |
| "99.646.234.875.495.327.686.473" |  |  |  |
| "00.927.234.572.289.123.934.865" |  |  |  |

* Thực hiện phép cộng chữ số hàng đơn vị.

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 0 |
| "99.646.234.875.495.327.686.473" | '3' | 3 |  |
| "00.927.234.572.289.123.934.865" | '5' | 5 |  |
|  |  | 8 |  |
| "8" | '8' |  | 0 |

* Thực hiện phép cộng chữ số hàng chục.

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 0 |
| "99.646.234.875.495.327.686.473" | '7' | 7 |  |
| "00.927.234.572.289.123.934.865" | '6' | 6 |  |
|  |  | 13 |  |
| "38" | '3' |  | 1 |

* Thực hiện phép cộng chữ số hàng trăm.

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "99.646.234.875.495.327.686.473" | '4' | 4 |  |
| "00.927.234.572.289.123.934.865" | '8' | 8 |  |
|  |  | 13 |  |
| "338" | '3' |  | 1 |

* Thực hiện phép cộng chữ số hàng ngàn.

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "99.646.234.875.495.327.686.473" | '6' | 6 |  |
| "00.927.234.572.289.123.934.865" | '4' | 4 |  |
|  |  | 11 |  |
| "1.338" | '1' |  | 1 |

* Thực hiện tiếp tục … ta được

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 0 |
| "99.646.234.875.495.327.686.473" | '6' | 6 |  |
| "00.927.234.572.289.123.934.865" | '9' | 9 |  |
|  |  | 15 |  |
| "573.469.447.784.451.621.338" | '5' |  | 1 |

* Thực hiện cộng chữ số hàng 22 ta được

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "99.646.234.875.495.327.686.473" | '9' | 9 |  |
| "00.927.234.572.289.123.934.865" | '0' | 0 |  |
|  |  | 10 |  |
| "0.573.469.447.784.451.621.338" | '0' |  | 1 |

* Thực hiện cộng chữ số hàng 23 ta được

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "99.646.234.875.495.327.686.473" | '9' | 9 |  |
| "00.927.234.572.289.123.934.865" | '0' | 0 |  |
|  |  | 10 |  |
| "00.573.469.447.784.451.621.338" | '0' |  | 1 |

* Cộng nhớ

|  | char | int | Nhớ |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "99.646.234.875.495.327.686.473" |  |  |  |
| "00.927.234.572.289.123.934.865" |  |  |  |
|  |  |  |  |
| "100.573.469.447.784.451.621.338" | '1' |  | 0 |

Khai báo hàm cộng hai số nguyên lớn:

1. BigNum addBigNum(BigNum, BigNum);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm addBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Định nghĩa hàm cộng hai số nguyên lớn:

1. BigNum addBigNum(BigNum x, BigNum y)
2. {
3. while (x.size() < y.size()) x = '0' + x;
4. while (x.size() > y.size()) y = '0' + y;
5. int carry = 0;
6. BigNum temp="";
7. for (int i = x.size() - 1; i >= 0; i--)
8. {
9. int xx = toNumber(x[i]);
10. int yy = toNumber(y[i]);
11. int ss = xx + yy + carry;
12. temp = toChar(ss % 10) + temp;
13. carry = ss / 10;
14. }
15. if (carry > 0)
16. temp = "1" + temp;
17. return temp;
18. }

* Viết với ngôn ngữ .

## Phép trừ hai số nguyên lớn

* Thực hiện phép trừ hai số nguyên lớn ngược lại với việc nhớ ở phép cộng hai số nguyên lớn. Trong phép trừ hai số nguyên lớn ta phải chú ý đến việc vay mượn từ hàng cao hơn.

Phép trừ hai số nguyên lớn:

* Thực hiện từ phải qua trái
* Phần mượn được mang sang trái.
* Hãy thực hiện hiện phép trừ hai số nguyên lớn sau:

|  |  |  |  |
| --- | --- | --- | --- |
| "10.646.234.875.495.327.686.473" |  |  |  |
| "927.234.572.289.123.934.865" |  |  |  |

* Thêm các ký tự 0 hợp lý để hai số nguyên lớn có độ dài bằng nhau.

|  |  |  |  |
| --- | --- | --- | --- |
| "10.646.234.875.495.327.686.473" |  |  |  |
| "00.927.234.572.289.123.934.865" |  |  |  |

* Thực hiện phép trừ chữ số hàng đơn vị.

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 0 |
| "10.646.234.875.495.327.686.473" | '3' | 3 |  |
| "00.927.234.572.289.123.934.865" | '5' | 5 |  |
|  |  | -2 |  |
|  |  | 8 | 1 |
| "8" | '8' |  | 1 |

* Thực hiện phép trừ chữ số hàng chục.

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "10.646.234.875.495.327.686.473" | '7' | 7 |  |
| "00.927.234.572.289.123.934.865" | '6' | 6 |  |
|  |  | 0 |  |
|  |  | 0 | 0 |
| "08" | '0' |  | 0 |

* Thực hiện phép trừ chữ số hàng trăm.

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 0 |
| "10.646.234.875.495.327.686.473" | '4' | 4 |  |
| "00.927.234.572.289.123.934.865" | '8' | 8 |  |
|  |  | -4 |  |
|  |  | 6 | 1 |
| "608" | '6' |  | 1 |

* Thực hiện phép trừ chữ số hàng ngàn.

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "10.646.234.875.495.327.686.473" | '6' | 6 |  |
| "00.927.234.572.289.123.934.865" | '4' | 4 |  |
|  |  | 1 |  |
|  |  | 1 | 0 |
| "1.608" | '1' |  | 0 |

* Thực hiện tiếp tục … ta được

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 0 |
| "10.646.234.875.495.327.686.473" | '6' | 6 |  |
| "00.927.234.572.289.123.934.865" | '9' | 9 |  |
|  |  | -3 |  |
|  |  | 7 | 1 |
| "719.000.303.206.203.751.608" | '7' |  | 1 |

* Thực hiện trừ chữ số hàng 22 ta được

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "10.646.234.875.495.327.686.473" | '0' | 0 |  |
| "00.927.234.572.289.123.934.865" | '0' | 0 |  |
|  |  | -1 |  |
|  |  | 9 | 1 |
| "9.719.000.303.206.203.751.608" | '9' |  | 1 |

* Thực hiện trừ chữ số hàng 23 ta được

|  | char | int | Mượn |
| --- | --- | --- | --- |
|  |  |  | 1 |
| "10.646.234.875.495.327.686.473" | '1' | 1 |  |
| "00.927.234.572.289.123.934.865" | '0' | 0 |  |
|  |  | 0 |  |
|  |  | 0 | 0 |
| "09.719.000.303.206.203.751.608" | '0' |  | 0 |

Khai báo hàm trừ hai số nguyên lớn:

1. BigNum subBigNum(BigNum, BigNum);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm subBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?
* Trong hàm trừ hai số nguyên dưới đây, chỉ xét trường hợp số nguyên lớn (tham số thứ nhất) trừ số nguyên lớn nhỏ hơn (tham số thứ hai).

Định nghĩa hàm trừ hai số nguyên lớn:

1. BigNum subBigNum(BigNum x, BigNum y)
2. {
3. while (x.size() < y.size()) x = '0' + x;
4. while (x.size() > y.size()) y = '0' + y;
5. int borrow = 0;
6. BigNum temp = "";
7. for (int i = x.size() - 1; i >= 0; i--)
8. {
9. int xx = toNumber(x[i]);
10. int yy = toNumber(y[i]);
11. int ss = xx - yy - borrow;
12. if (ss < 0)
13. {
14. ss = ss + 10;
15. borrow = 1;
16. }
17. else
18. borrow = 0;
19. temp = toChar(ss) + temp;
20. }
21. while (temp.size() > 1 && temp[0] == '0')
22. temp.erase(0, 1);
23. return temp;
24. }

Lưu ý: hàm trừ hai số nguyên trên, chỉ xét trường hợp số nguyên lớn (tham số thứ nhất) trừ số nguyên lớn nhỏ hơn (tham số thứ hai).

* Viết với ngôn ngữ .

## Phép nhân một số nguyên lớn với một số nguyên nhỏ

* Số nguyên nhỏ là số nguyên có kiểu dữ liệu là long (kiểu long long khi nhân có thể gây tràn số).
* Hãy thực hiện hiện phép nhân một số nguyên lớn với một số nguyên nhỏ như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| "86.473" |  |  |  |
| 4.865 |  |  |  |
|  |  |  |  |
| "" |  |  |  |

* Thực hiện phép nhân chữ số hàng đơn vị của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Nhớ |
|  |  |  | 0 |
| "86.473" | '3' | 3 |  |
| 4.865 |  |  |  |
|  |  | 14.595 |  |
|  | '5' |  | 1.459 |
| "5" |  |  |  |

* Thực hiện phép nhân chữ số hàng chục của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Nhớ |
|  |  |  | 1,459 |
| "86.473" | '7' | 7 |  |
| 4.865 |  |  |  |
|  |  | 35.514 |  |
|  | '4' |  | 3.551 |
| "45" |  |  |  |

* Thực hiện phép nhân chữ số hàng trăm của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Nhớ |
|  |  |  | 3.551 |
| "86.473" | '4' | 4 |  |
| 4.865 |  |  |  |
|  |  | 23.011 |  |
|  | '1' |  | 2.301 |
| "145" |  |  |  |

* Thực hiện phép nhân chữ số hàng ngàn của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Nhớ |
|  |  |  | 2.301 |
| "86.473" | '6' | 6 |  |
| 4.865 |  |  |  |
|  |  | 31.491 |  |
|  | '1' |  | 3.149 |
| "1.145" |  |  |  |

* Thực hiện phép nhân chữ số hàng chục ngàn của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Nhớ |
|  |  |  | 3.149 |
| "86.473" | '8' | 8 |  |
| 4.865 |  |  |  |
|  |  | 42.069 |  |
|  | '9' |  | 4.206 |
| "91.145" |  |  |  |

* Cộng nhớ

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Nhớ |
|  |  |  | 4.206 |
| "86.473" |  |  |  |
| 4.865 |  |  |  |
|  |  |  |  |
| "420.691.145" |  |  |  |

Khai báo hàm nhân một số nguyên lớn với một số nguyên nhỏ:

1. BigNum mulBigNum(BigNum, long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm mulBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Định nghĩa hàm nhân một số nguyên lớn với một số nguyên nhỏ:

1. BigNum mulBigNum(BigNum x, long y)
2. {
3. long carry = 0;
4. BigNum temp = "";
5. for (int i = x.size() - 1; i >= 0; i--)
6. {
7. int xx = toNumber(x[i]);
8. long ss = xx \* y + carry;
9. temp = toChar(ss % 10) + temp;
10. carry = ss / 10;
11. }
12. if (carry > 0)
13. temp = to\_string(carry) + temp;
14. return temp;
15. }

* Viết với ngôn ngữ .

## Phép nhân hai số nguyên lớn

* Hãy thực hiện hiện phép nhân hai số nguyên lớn sau:

|  |  |  |  |
| --- | --- | --- | --- |
| "86.473" |  |  |  |
| "4.865" |  |  |  |
|  |  |  |  |
| "" |  |  |  |

* Thực hiện phép nhân chữ số hàng đơn vị của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Zero |
| "" |  |  | "" |
| "86.473" | '3' | 3 |  |
| "4.865" |  |  |  |
|  |  | "14.595" |  |
|  |  | "14.595" |  |
| "14.595" |  |  | "0" |

* Thực hiện phép nhân chữ số hàng chục của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Zero |
| "14.595" |  |  | "0" |
| "86.473" | '7' | 7 |  |
| "4.865" |  |  |  |
|  |  | "34.055" |  |
|  |  | "340.550" |  |
| "355.145" |  |  | "00" |

* Thực hiện phép nhân chữ số hàng trăm của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Zero |
| "355.145" |  |  | "00" |
| "86.473" | '4' | 4 |  |
| "4.865" |  |  |  |
|  |  | "19.460" |  |
|  |  | "1.946.000" |  |
| "2.301.145" |  |  | "000" |

* Thực hiện phép nhân chữ số hàng ngàn của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Zero |
| "2.301.145" |  |  | "000" |
| "86.473" | '6' | 6 |  |
| "4.865" |  |  |  |
|  |  | "29.190" |  |
|  |  | "29.190.000" |  |
| "31.491.145" |  |  | "0000" |

* Thực hiện phép nhân chữ số hàng chục ngàn của số nguyên lớn với số nguyên nhỏ:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Zero |
| "31.491.145" |  |  | "0000" |
| "86.473" | '8' | 8 |  |
| "4.865" |  |  |  |
|  |  | "38.920" |  |
|  |  | "389.200.000" |  |
| "420.691.145" |  |  | "00000" |

Khai báo hàm nhân hai số nguyên lớn:

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm mulBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

1. BigNum mulBigNum(BigNum, BigNum);

Định nghĩa hàm nhân hai số nguyên lớn:

1. BigNum mulBigNum(BigNum x, BigNum y)
2. {
3. BigNum temp = "";
4. BigNum zero = "";
5. for (int i = x.size() - 1; i >= 0; i--)
6. {
7. long xx = toNumber(x[i]);
8. BigNum ss = mulBigNum(y, xx);
9. ss = ss + zero;
10. temp = addBigNum(temp, ss);
11. zero = zero + '0';
12. }
13. return temp;
14. }

* Viết với ngôn ngữ .

## Phép chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ

Minh họa cho việc thực hiện hiện phép chia lấy nguyên số nguyên lớn cho số nguyên nhỏ :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 8 | 4 | 3 | 1 | 7 |  | 4 | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  | 0 |  |  |  |  |  |  |
| 2 | 8 |  |  |  |  |  | 0 | 0 |  |  |  |  |  |
| 2 | 8 | 4 |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 4 | 6 |  |  |  |  | 0 | 0 | 6 |  |  |  |  |
|  | 3 | 8 |  |  |  |  |  |  |  |  |  |  |  |
|  | 3 | 8 | 3 |  |  |  |  |  |  |  |  |  |  |
|  | 3 | 6 | 9 |  |  |  | 0 | 0 | 6 | 9 |  |  |  |
|  |  | 1 | 4 |  |  |  |  |  |  |  |  |  |  |
|  |  | 1 | 4 | 1 |  |  |  |  |  |  |  |  |  |
|  |  | 1 | 2 | 3 |  |  | 0 | 0 | 6 | 9 | 3 |  |  |
|  |  |  | 1 | 8 |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 8 | 7 |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 6 | 4 |  | 0 | 0 | 6 | 9 | 3 | 4 |  |
|  |  |  |  | 2 | 3 |  |  |  |  |  |  |  |  |

Khai báo hàm chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ:

1. BigNum divBigNum(BigNum, long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm divBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Định nghĩa hàm chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ:

1. BigNum divBigNum(BigNum x, long y)
2. {
3. BigNum temp = "";
4. long hold = 0;
5. long ss = 0;
6. for (int i = 0; i < x.size(); i++)
7. {
8. hold = hold \* 10 + toNumber(x[i]);
9. ss = hold / y;
10. temp = temp + toChar(ss);
11. hold = hold % y;
12. }
13. while (temp.size() > 1 && temp[0] == '0')
14. temp.erase(0, 1);
15. return temp;
16. }

* Viết với ngôn ngữ .

## Phép chia lấy dư (mod) một số nguyên lớn với một số nguyên nhỏ

Minh họa cho việc thực hiện hiện phép chia lấy dư số nguyên lớn cho số nguyên nhỏ :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 8 | 4 | 3 | 1 | 7 |  | 4 | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  | 0 |  |  |  |  |  |  |
| 2 | 8 |  |  |  |  |  | 0 | 0 |  |  |  |  |  |
| 2 | 8 | 4 |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 4 | 6 |  |  |  |  | 0 | 0 | 6 |  |  |  |  |
|  | 3 | 8 |  |  |  |  |  |  |  |  |  |  |  |
|  | 3 | 8 | 3 |  |  |  |  |  |  |  |  |  |  |
|  | 3 | 6 | 9 |  |  |  | 0 | 0 | 6 | 9 |  |  |  |
|  |  | 1 | 4 |  |  |  |  |  |  |  |  |  |  |
|  |  | 1 | 4 | 1 |  |  |  |  |  |  |  |  |  |
|  |  | 1 | 2 | 3 |  |  | 0 | 0 | 6 | 9 | 3 |  |  |
|  |  |  | 1 | 8 |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 8 | 7 |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 6 | 4 |  | 0 | 0 | 6 | 9 | 3 | 4 |  |
|  |  |  |  | 2 | 3 |  |  |  |  |  |  |  |  |

Khai báo hàm chia lấy dư (mod) một số nguyên lớn với một số nguyên nhỏ:

1. long modBigNum(BigNum, long);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm modBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Nhắc lại định nghĩa hàm chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ:

1. BigNum divBigNum(BigNum x, long y)
2. {
3. BigNum temp = "";
4. long hold = 0;
5. long ss = 0;
6. for (int i = 0; i < x.size(); i++)
7. {
8. hold = hold \* 10 + toNumber(x[i]);
9. ss = hold / y;
10. temp = temp + toChar(ss);
11. hold = hold % y;
12. }
13. while (temp.size() > 1 && temp[0] == '0')
14. temp.erase(0, 1);
15. return temp;
16. }

Định nghĩa hàm phép chia lấy dư một số nguyên lớn với một số nguyên nhỏ bằng cách lược bỏ một số câu lệnh trong định nghĩa hàm chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ

1. long modBigNum(BigNum x, long y)
2. {
3. long hold = 0;
4. for (int i = 0; i < x.size(); i++)
5. {
6. hold = hold \* 10 + toNumber(x[i]);
7. hold = hold % y;
8. }
9. return hold;
10. }

* Viết với ngôn ngữ .

Định nghĩa hàm (viết lại hàm trên gọn hơn nữa) chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên nhỏ:

1. long modBigNum(BigNum x, long y)
2. {
3. long hold = 0;
4. for (int i = 0; i < x.size(); i++)
5. hold = (hold \* 10 + toNumber(x[i])) % y;
6. return hold;
7. }

* Viết với ngôn ngữ .

## Phép chia lấy thương nguyên (div) hai số nguyên lớn

Minh họa cho việc thực hiện hiện phép chia lấy thương nguyên số nguyên lớn cho số nguyên lớn :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 8 | 4 | 3 | 1 | 7 |  | 4 | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  | 0 |  |  |  |  |  |  |
| 2 | 8 |  |  |  |  |  | 0 | 0 |  |  |  |  |  |
| 2 | 8 | 4 |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 4 | 6 |  |  |  |  | 0 | 0 | 6 |  |  |  |  |
|  | 3 | 8 |  |  |  |  |  |  |  |  |  |  |  |
|  | 3 | 8 | 3 |  |  |  |  |  |  |  |  |  |  |
|  | 3 | 6 | 9 |  |  |  | 0 | 0 | 6 | 9 |  |  |  |
|  |  | 1 | 4 |  |  |  |  |  |  |  |  |  |  |
|  |  | 1 | 4 | 1 |  |  |  |  |  |  |  |  |  |
|  |  | 1 | 2 | 3 |  |  | 0 | 0 | 6 | 9 | 3 |  |  |
|  |  |  | 1 | 8 |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 8 | 7 |  |  |  |  |  |  |  |  |
|  |  |  | 1 | 6 | 4 |  | 0 | 0 | 6 | 9 | 3 | 4 |  |
|  |  |  |  | 2 | 3 |  |  |  |  |  |  |  |  |

Khai báo hàm chia lấy thương nguyên (div) hai số nguyên lớn:

1. BigNum divBigNum(BigNum, BigNum);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm divBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Định nghĩa hàm chia lấy thương nguyên (div) hai số nguyên lớn:

1. BigNum divBigNum(BigNum x, BigNum y)
2. {
3. BigNum kY[11];
4. kY[0] = "0";
5. for (int k = 1; k <= 10; k++)
6. kY[k] = addBigNum(kY[k - 1], y);
8. BigNum temp = "";
9. BigNum hold = "";
10. for (int i = 0; i < x.size(); i++)
11. {
12. hold = hold + x[i];
13. int k = 0;
14. while (compareBigNum(hold,kY[k+1])!=-1)
15. k++;
16. temp = temp + toChar(k);
17. hold = subBigNum(hold, kY[k]);
18. }
19. while (temp.size() > 1 && temp[0] == '0')
20. temp.erase(0, 1);
21. return temp;
22. }

* Viết với ngôn ngữ .

## Phép chia lấy dư (mod) hai số nguyên lớn

Khai báo hàm chia lấy dư (mod) hai số nguyên lớn:

1. BigNum modBigNum(BigNum, BigNum);

* Tên hàm?
* Kiểu dữ liệu trả về?
* Mấy tham số đầu vào?
* Tham số thứ nhất kiểu dữ liệu là gì?
* Tham số thứ hai kiểu dữ liệu là gì?
* Loại tham số thứ nhất?
* Loại tham số thứ hai?
* Khi gọi hàm modBigNum ta có giá trị của tham số thứ nhất và tham số thứ hai chưa?

Nhắc lại định nghĩa hàm chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên lớn:

1. BigNum divBigNum(BigNum x, BigNum y)
2. {
3. BigNum kY[11];
4. kY[0] = "0";
5. for (int k = 1; k <= 10; k++)
6. kY[k] = addBigNum(kY[k - 1], y);
8. BigNum temp = "";
9. BigNum hold = "";
10. for (int i = 0; i < x.size(); i++)
11. {
12. hold = hold + x[i];
13. int k = 0;
14. while (compareBigNum(hold,kY[k+1])!=-1)
15. k++;
16. temp = temp + toChar(k);
17. hold = subBigNum(hold, kY[k]);
18. }
19. while (temp.size() > 1 && temp[0] == '0')
20. temp.erase(0, 1);
21. return temp;
22. }

Định nghĩa hàm phép chia lấy dư một số nguyên lớn với một số nguyên lớn bằng cách lược bỏ một số câu lệnh trong định nghĩa hàm chia lấy thương nguyên (div) một số nguyên lớn với một số nguyên lớn.

1. BigNum modBigNum(BigNum x, BigNum y)
2. {
3. BigNum kY[11];
4. kY[0] = "0";
5. for (int i = 1; i <= 10; i++)
6. kY[i] = addBigNum(kY[i - 1], y);
7. BigNum hold = "";
8. for (int i = 0; i < x.size(); i++)
9. {
10. hold = hold + x[i];
11. int k = 0;
12. while(compareBigNum(hold,kY[k+1])>=0)
13. k++;
14. hold = subBigNum(hold, kY[k]);
15. }
16. return hold;
17. }

* Viết với ngôn ngữ .

## Tính số Fibonacci thứ

Khai báo hàm tính số hạng thứ :

1. BigNum addBigNum(BigNum, BigNum);
2. BigNum BigFibonacci(long);

Định nghĩa hàm tính số hạng thứ :

1. BigNum BigFibonacci(long n)
2. {
3. BigNum Ftt = "1";
4. BigNum Ft = "1";
5. long i = 2;
6. BigNum Fhh;
7. while (i <= n)
8. {
9. Fhh = addBigNum(Ft,Ftt);
10. i++;
11. Ftt = Ft;
12. Ft = Fhh;
13. }
14. return Fhh;
15. }

* Viết với ngôn ngữ .

1. Viết chương trình tính số hạng thứ .

* Viết với ngôn ngữ .

1. #include <iostream>
2. using namespace std;
3. typedef string BigNum;
4. int toNumber(char);
5. char toChar(int);
6. BigNum addBigNum(BigNum, BigNum);
7. BigNum BigFibonacci(long);
8. int main()
9. {
10. long n;
11. cout << "Nhap n: ";
12. cin >> n;
13. cout << "So hang fibo " << n << " : ";
14. cout << << BigFibonacci(n);
15. cout << "\n\n\nKet thuc!!!!";
16. return 1;
17. }

* Viết với ngôn ngữ .

## Tính giai thừa thứ n

Khai báo hàm tính giai thừa thứ :

1. BigNum BigFactorial(long);

Định nghĩa hàm tính giai thừa thứ :

1. BigNum BigFactorial(long n)
2. {
3. BigNum T = "1";
4. for (long i = 1; i <= n; i++)
5. T = mulBigNum(T, i);
6. return T;
7. }

* Viết với ngôn ngữ .

1. Viết chương trình tính .
2. #include <iostream>
3. #include <string>
4. using namespace std;
5. typedef string BigNum;
6. int toNumber(char);
7. char toChar(int);
8. BigNum mulBigNum(BigNum, long);
9. BigNum BigFactorial(long);
10. int main()
11. {
12. long n;
13. cout << "Nhap n: ";
14. cin >> n;
15. cout << "So hang fibo " << n << " : ";
16. cout << BigFactorial(n);
17. cout << "\n\n\nKet thuc!!!!";
18. return 1;
19. }
20. ...

* Viết với ngôn ngữ .

## Tính số hạng Catalan thứ

Khai báo hàm tính số hạng thứ :

1. BigNum Catalan(long);

Định nghĩa hàm tính số hạng thứ :

1. BigNum Catalan(long n)
2. {
3. BigNum a = BigFactorial(2 \* n);
4. BigNum b = BigFactorial(n);
5. BigNum c = BigFactorial(n + 1);
6. return divBigNum(divBigNum(a, b), c);
7. }

* Viết với ngôn ngữ .

1. Viết chương trình tính số hạng thứ .
2. #include <iostream>
3. #include <string>
4. using namespace std;
5. typedef string BigNum;
6. int toNumber(char);
7. char toChar(int);
8. int compareBigNum(BigNum, BigNum);
9. BigNum addBigNum(BigNum, BigNum);
10. BigNum subBigNum(BigNum, BigNum);
11. BigNum mulBigNum(BigNum, long);
12. BigNum divBigNum(BigNum, BigNum);
13. BigNum BigFactorial(long);
14. BigNum BigCatalan(long);
15. int main()
16. {
17. long n;
18. cout << "Nhap n: ";
19. cin >> n;
20. cout << "So hang catalan " << n;
21. cout << "! : " << BigCatalan(n);
22. cout << "\n\n\nKet thuc!!!!";
23. return 1;
24. }

* Viết với ngôn ngữ .

## Bài tập