apc.umt@gmail.com |
https://github.com/GrootTheDeveloper/Applied-Programming-Club

APC ROAD TO OLP

Tài liệu ôn tập Competitive Programming dành cho thành viên Câu lạc bộ Lập trình ứng dụng

Câu lạc bộ Lập trình ứng dụng - Applied Programming Club

Câu lạc bộ trực thuộc UMT - Khoa Công nghệ

Ngày 12 tháng 1 năm 2026

Mục lục

LỜI NÓI ĐẦU

Trong thời đại công nghệ số, lập trình không chỉ là một kỹ năng quan trọng mà còn là cánh cửa mở ra vô vàn cơ hội phát triển cho sinh viên trong lĩnh vực công nghệ thông tin và khoa học máy tính. Đặc biệt, với các cuộc thi lập trình như **Olympic Tin học Sinh viên Việt Nam (OLP)**, **International Collegiate Programming Contest (ICPC)** hay các kỳ thi trực tuyến toàn cầu, lập trình thi đấu (*Competitive Programming - CP*) đã trở thành môi trường tuyệt vời để sinh viên rèn luyện tư duy logic, kỹ năng phân tích và khả năng giải quyết vấn đề.

Với khát vọng tạo dựng một cộng đồng học thuật năng động tại Trường Đại học Quản lý và Công nghệ TP.HCM (UMT) và hỗ trợ sinh viên UMT tiếp cận *Competitive Programming* từ con số 0, **Câu lạc bộ Lập trình ứng dụng (APC)** trực thuộc UMT - Khoa Công nghệ đã biên soạn bộ tài liệu này với mong muốn đồng hành cùng sinh viên từ những bước đi đầu tiên trong hành trình lập trình thi đấu.

Tài liệu được thiết kế với ba định hướng chính:

- **Dễ tiếp cận cho người mới bắt đầu:** Nội dung bắt đầu từ những kiến thức lập trình cơ bản nhất, không yêu cầu nền tảng trước, giúp sinh viên học từ con số 0.
- **Hệ thống kiến thức nền tảng và nâng cao:** Bao quát các chủ đề từ **Lập trình cơ bản**, **Cấu trúc dữ liệu** đến **Thuật toán**, tạo nền móng vững chắc cho việc tham gia lập trình thi đấu và phát triển kỹ năng lập trình chuyên sâu.
- **Hướng đến thực hành và thi đấu:** Cung cấp bài tập và phân tích chiến lược giải bài, giúp sinh viên không chỉ học lý thuyết mà còn rèn luyện khả năng tư duy và phản xạ trong môi trường thi đấu thực tế.

Tài liệu này không chỉ là nguồn học tập, mà còn là cầu nối gắn kết các thành viên trong CLB APC, tạo ra một cộng đồng học thuật năng động, nơi các bạn sinh viên cùng nhau học hỏi, trao đổi và tiến bộ. Nhóm biên soạn hy vọng rằng với sự đồng hành của tài liệu này, mỗi bạn sinh viên đều có thể từng bước vươn lên, từ người mới bắt đầu đến thí sinh tự tin tham gia các kỳ thi lập trình, góp phần nâng cao vị thế của UMT trên các sân chơi học thuật trong và ngoài nước.

Trong quá trình biên soạn, nhóm đã nỗ lực hết mình để đảm bảo tính chính xác và tính thực tiễn. Tuy nhiên, do phạm vi kiến thức rộng lớn, khó tránh khỏi những thiếu sót. Chúng tôi rất mong nhận được những ý kiến đóng góp từ bạn đọc để tài liệu ngày càng hoàn thiện hơn.

TP. Hồ Chí Minh, Ngày 12 tháng 1 năm 2026

Nhóm biên soạn tài liệu APC

CHƯƠNG 1

TỔNG QUAN

Người soạn: Đặng Phúc An Khang – Khóa 2 ngành Công nghệ Thông tin
Tài liệu tham khảo chính: *Competitive Programmer's Handbook* – Antti Laaksonen

Contents

1.1 Chương trình đánh giá & Xếp hạng

Chương trình áp dụng cơ chế **tính điểm minh bạch** để theo dõi tiến độ, khuyến khích sự chủ động và duy trì kỷ luật học tập trong suốt lộ trình. **Điểm tổng** của mỗi học viên được tính theo công thức:

$$\text{Điểm tổng} = \text{Điểm cộng} - \text{Điểm trừ}.$$

Bảng xếp hạng (BXH) được cập nhật định kỳ dựa trên điểm tổng và là căn cứ để xét giải thưởng.

1.1.1 Cơ cấu điểm cộng

1.1.1.1 Điểm bài tập

Mỗi bài tập hoàn thành sẽ được cộng điểm theo độ khó như Bảng 1.1.

Bảng 1.1: Điểm cơ bản theo loại bài tập

Loại bài	**Điểm/Bài**
Easy	+2
Medium	+4
Hard	+7

Ngoài điểm cơ bản, học viên có thể nhận thêm điểm thưởng khi đạt các tiêu chí trong Bảng 1.2.

Bảng 1.2: Điểm thưởng khi nộp bài

Điểm cộng	**Điểm**
Hoàn thành trong 24h sau khi ra bài	+1
Viết lời giải bằng LaTeX	+3

1.1.1.2 Điểm cuộc thi nội bộ

Cứ **mỗi 4 tuần** học sẽ tổ chức **01 cuộc thi nội bộ**. Điểm cuộc thi tính theo thứ hạng:

- Xếp hạng 1: +12 điểm.
- Xếp hạng 2: +11 điểm.
- ...
- Xếp hạng 12: +1 điểm.
- Không tham gia: 0 điểm.

1.1.1.3 Điểm cuộc thi ngoài (Codeforces, VNOI, ...)

Học viên tham gia các cuộc thi bên ngoài (có minh chứng hợp lệ) sẽ được:

- Tham gia: +2 điểm.

1.1.2 Cơ cấu điểm trừ

1.1.2.1 Vi phạm AI/Sao chép

Để đảm bảo tính trung thực và chất lượng rèn luyện, các vi phạm sau sẽ bị xử lý nghiêm:

1. **Dùng AI để nộp bài**
   - Bài đó: 0 điểm.
   - Trừ thêm: 10 điểm.
2. **Tái phạm (lần 2)**
   - Cảnh cáo công khai.
   - Không được tham gia xét giải.

1.1.2.2 Vi phạm chuyên cần

Các hành vi vi phạm chuyên cần bị trừ điểm như sau (Bảng 1.3).

Bảng 1.3: Điểm trừ do vi phạm chuyên cần

Hành vi	**Điểm**
Nghỉ học không lý do	-5
Không nộp hơn 50% bài tập tuần (bài tập bắt buộc)	-5
Bỏ tham gia cuộc thi nội bộ lớp học không lý do	-5

1.1.3 Nguyên tắc xếp hạng và xét giải

- BXH dựa trên **điểm tổng** và được cập nhật theo chu kỳ của chương trình.
- Học viên có vi phạm **AI/Sao chép tái phạm** sẽ **không đủ điều kiện xét giải**, dù điểm số cao.
- Kết quả cuối cùng được chốt tại thời điểm tổng kết chương trình.

1.1.4 Cơ cấu giải thưởng

Giải thưởng được trao nhằm ghi nhận cả **thành tích** lẫn **sự bền bỉ** trong quá trình học (Bảng 1.4).

Bảng 1.4: Cơ cấu giải thưởng

Giải	**Điều kiện**
OLP Champion	Tổng điểm cao nhất BXH
Top Performer (02 giải)	Xếp hạng Top 2–3 tổng điểm
Consistency Award	Hoàn thành nhiều bài tập nhất
Most Improved	So sánh tổng điểm sau 2 tuần đầu v trình; ai tăng nhiều điểm nhất thắng

1.2 Giới thiệu về Lập trình thi đấu

1.2.1 Lập trình thi đấu là gì?

Lập trình thi đấu (Competitive Programming - CP) là hình thức giải các bài toán thuật toán (làm bài trên máy tính) trong một khoảng thời gian hữu hạn (thường từ 1,5 đến 5 giờ), dưới những *ràng buộc chặt chẽ* về thời gian chạy và bộ nhớ. Thí sinh (cá nhân hoặc đội) viết chương trình nhằm:

- **Đọc** dữ liệu từ bàn phím/file và **ghi** kết quả ra màn hình/file.
- **Được chấm tự động** trên bộ test ẩn bởi hệ thống *Online Judge* (OJ) hoặc chấm offline với các kỳ thi như Học sinh giỏi Quốc gia (HSGQG).

- **Được đánh giá** theo các tiêu chí chính: *đúng* (correctness), *nhanh* (time complexity/performance), *tiết kiệm bộ nhớ* (space usage), đôi khi có *điểm từng phần* (partial score).

Lập trình thi đấu vì thế vừa là một hình thức thi, vừa là môi trường rèn luyện tư duy thuật toán và kỹ năng lập trình trong điều kiện áp lực về thời gian và tài nguyên.

1.2.2 Thiết kế thuật toán trong lập trình thi đấu

Lập trình thi đấu kết hợp hai nội dung chính:

- **Thiết kế thuật toán**
- **Hiện thực (cài đặt) thuật toán**

Việc thiết kế thuật toán bao gồm quá trình giải quyết bài toán và tư duy toán học. Người học cần có kỹ năng phân tích bài toán và khả năng tìm ra lời giải một cách sáng tạo. Một thuật toán giải quyết bài toán phải vừa đúng vừa hiệu quả, và cốt lõi của nhiều bài toán thường nằm ở việc xây dựng được một thuật toán có độ phức tạp tối ưu.

Kiến thức lý thuyết về thuật toán đóng vai trò quan trọng đối với lập trình viên thi đấu. Thông thường, lời giải cho một bài toán là sự kết hợp giữa các kỹ thuật đã được biết đến và những ý tưởng mới. Các kỹ thuật xuất hiện trong lập trình thi đấu cũng chính là nền tảng cho các nghiên cứu khoa học trong lĩnh vực thuật toán.

1.2.3 Hiện thực thuật toán và phong cách lập trình

Việc hiện thực thuật toán đòi hỏi kỹ năng lập trình tốt. Trong lập trình thi đấu, các lời giải được đánh giá thông qua việc kiểm tra chương trình đã cài đặt bằng một tập bộ test do hệ thống cung cấp. Do đó, không chỉ ý tưởng thuật toán cần đúng, mà phần cài đặt cũng phải hoàn toàn chính xác; một lỗi nhỏ trong cài đặt cũng có thể dẫn tới kết quả *Wrong Answer* hoặc *Runtime Error*.

Phong cách lập trình trong các kỳ thi thường cần **đơn giản và súc tích**. Chương trình phải được viết nhanh do thời gian hạn chế, đồng thời đủ rõ ràng để dễ dàng sửa lỗi nếu cần. Khác với kỹ nghệ phần mềm truyền thống, các chương trình trong lập trình thi đấu thường ngắn (thường không quá vài trăm dòng mã) và không cần bảo trì sau khi cuộc thi kết thúc.

1.2.4 Một số thuật ngữ và *verdict* thường gặp

Trong quá trình nộp bài trên các hệ thống chấm tự động, lập trình viên thi đấu thường gặp các *verdict* sau:

- `AC` (Accepted): Kết quả đúng trên tất cả test.
- `WA` (Wrong Answer): Sai kết quả trên ít nhất một test.
- `TLE` (Time Limit Exceeded): Vượt quá giới hạn thời gian cho phép.
- `MLE` (Memory Limit Exceeded): Vượt quá giới hạn bộ nhớ.
- `RE` (Runtime Error): Lỗi trong quá trình chạy (ví dụ chia cho 0, truy cập ngoài mảng,...).
- `CE` (Compilation Error): Lỗi biên dịch, chương trình không biên dịch được.

Những thuật ngữ này là một phần không thể tách rời của lập trình thi đấu và giúp thí sinh hiểu rõ hơn lý do bài làm của mình chưa được chấp nhận (nếu không được `AC`).

1.3 Ngôn ngữ lập trình

Hiện nay, các ngôn ngữ lập trình được sử dụng phổ biến nhất trong lập trình thi đấu là **C++**, **Python** và **Java**. Theo thống kê từ cuộc thi Google Code Jam năm 2017, trong số khoảng 3.000 thí sinh có thứ hạng cao nhất,

khoảng 79% sử dụng C++, 16% sử dụng Python và 8% sử dụng Java. Ngoài ra, một số thí sinh còn sử dụng kết hợp nhiều ngôn ngữ khác nhau tùy theo từng bài toán.

Nhiều lập trình viên cho rằng C++ là lựa chọn tối ưu nhất cho lập trình thi đấu, và trên thực tế, C++ gần như luôn được hỗ trợ trong tất cả các hệ thống chấm bài. Ưu điểm lớn nhất của C++ là hiệu năng cao, khả năng kiểm soát bộ nhớ tốt, cùng với thư viện chuẩn phong phú, cung cấp nhiều cấu trúc dữ liệu và thuật toán hiệu quả. Những yếu tố này giúp C++ đặc biệt phù hợp với các bài toán có ràng buộc nghiêm ngặt về thời gian và bộ nhớ.

Tuy nhiên, việc thành thạo nhiều ngôn ngữ lập trình và hiểu rõ điểm mạnh của từng ngôn ngữ cũng rất quan trọng. Chẳng hạn, trong các bài toán yêu cầu xử lý số nguyên rất lớn, Python có thể là một lựa chọn thuận lợi nhờ hỗ trợ sẵn các phép toán trên số nguyên có độ chính xác cao. Dù vậy, hầu hết các bài toán trong lập trình thi đấu đều được thiết kế sao cho việc sử dụng một ngôn ngữ lập trình cụ thể không tạo ra lợi thế không công bằng so với các ngôn ngữ khác.

Trong tài liệu này, tất cả các ví dụ minh họa đều được viết bằng **C++** và thường xuyên sử dụng các cấu trúc dữ liệu và thuật toán trong thư viện chuẩn. Các chương trình tuân theo chuẩn **C++20**, vốn đã được hỗ trợ rộng rãi trong hầu hết các kỳ thi lập trình hiện nay. Đối với người học chưa quen với C++, đây là thời điểm thích hợp để bắt đầu làm quen và rèn luyện với ngôn ngữ này.

1.3.1 Mẫu code C++ trong lập trình thi đấu

Một mẫu chương trình C++ cơ bản thường được sử dụng trong lập trình thi đấu có dạng như sau:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    // solution comes here
}
```

Dòng `#include <bits/stdc++.h>` là một đặc điểm của trình biên dịch `g++`, cho phép nạp toàn bộ thư viện chuẩn của C++ chỉ trong một dòng lệnh. Nhờ đó, người lập trình không cần phải lần lượt khai báo các thư viện như `iostream`, `vector` hay `algorithm`, vì tất cả đã được bao gồm sẵn.

Dòng lệnh `using namespace std;` cho phép sử dụng trực tiếp các lớp và hàm trong thư viện chuẩn. Nếu không có dòng lệnh này, người lập trình sẽ phải viết đầy đủ tiền tố `std::`, ví dụ như `std::cout` thay vì chỉ cần `cout`.

Chương trình có thể được biên dịch bằng lệnh sau:

```
g++ -std=c++20 -O2 -Wall test.cpp -o test
```

Lệnh trên tạo ra tệp thực thi `test` từ mã nguồn `test.cpp`. Trình biên dịch sử dụng chuẩn C++20 (`-std=c++20`), tối ưu hóa chương trình (`-O2`) và hiển thị các cảnh báo liên quan đến lỗi tiềm ẩn trong mã nguồn (`-Wall`).

Sau khi biên dịch được tệp thực thi `test`, để chạy chương trình, ta sử dụng câu lệnh:

```
./test
```

1.4 Các cuộc thi và tài nguyên luyện thi

Lập trình thi đấu không chỉ gắn liền với việc rèn luyện thuật toán, mà còn gắn với các kỳ thi ở nhiều cấp độ khác nhau, từ học sinh phổ thông đến sinh viên đại học và cộng đồng trực tuyến. Phần này giới thiệu các cuộc thi tiêu biểu và những tài nguyên quan trọng phục vụ cho việc học và luyện tập.

1.4.1 IOI – International Olympiad in Informatics

International Olympiad in Informatics (IOI) là kỳ thi Olympic Tin học quốc tế được tổ chức hằng năm dành cho học sinh trung học phổ thông. Mỗi quốc gia được cử một đội gồm **4 thí sinh** tham dự. Thông thường, IOI có khoảng **300 thí sinh** đến từ hơn **80 quốc gia**.

Kỳ thi IOI gồm **2 ngày thi**, mỗi ngày kéo dài **5 giờ**. Trong mỗi ngày, thí sinh phải giải **3 bài toán thuật toán** với độ khó khác nhau. Các bài toán thường được chia thành nhiều **subtask**, mỗi subtask có số điểm riêng, cho phép thí sinh nhận điểm một phần.

Mặc dù thí sinh được tuyển chọn theo đội tuyển quốc gia, nhưng tại IOI, các thí sinh **thi đấu cá nhân**.

Nội dung ra đề của IOI được quy định trong *IOI Syllabus*, bao gồm các chủ đề thuật toán và cấu trúc dữ liệu cốt lõi. Hầu hết các chủ đề trong chương trình IOI đều là nền tảng của lập trình thi đấu hiện đại.

Trước khi tham dự IOI, thí sinh thường trải qua các kỳ thi chọn đội tuyển quốc gia, cũng như các kỳ thi khu vực như:

- Baltic Olympiad in Informatics (BOI)
- Central European Olympiad in Informatics (CEOI)
- Asia-Pacific Informatics Olympiad (APIO)

Ngoài ra, một số quốc gia tổ chức các kỳ thi trực tuyến để luyện tập cho IOI, ví dụ như:

- Croatian Open Competition in Informatics
- USA Computing Olympiad (USACO)

1.4.2 ICPC – International Collegiate Programming Contest

International Collegiate Programming Contest (ICPC) là kỳ thi lập trình quốc tế dành cho sinh viên đại học. Khác với IOI, ICPC là **cuộc thi theo đội**, mỗi đội gồm **3 sinh viên** và chỉ được sử dụng **1 máy tính** trong suốt thời gian thi.

Một cuộc thi ICPC tiêu chuẩn kéo dài **5 giờ**, trong đó mỗi đội cần giải khoảng **8–12 bài toán thuật toán**. Một bài toán chỉ được chấp nhận khi lời giải đúng **toàn bộ bộ test** và chạy hiệu quả trong giới hạn thời gian.

ICPC gồm nhiều vòng:

- Vòng miền (Nam/Trung/Bắc)
- Vòng quốc gia (National)
- Vòng khu vực (Regional)
- Chung kết thế giới (World Finals)

Mặc dù có hàng chục nghìn đội tham gia ICPC mỗi năm, số lượng suất vào chung kết thế giới là rất hạn chế. Do đó, chỉ cần lọt vào vòng World Finals đã được xem là một thành tích lớn.

Trong ICPC, bảng xếp hạng được công bố theo thời gian thực, nhưng **giờ cuối cùng** của cuộc thi thường bị *đóng băng* (frozen scoreboard), khiến kết quả cuối cùng chỉ được công bố sau khi kết thúc.

So với IOI, phạm vi kiến thức trong ICPC rộng hơn, đòi hỏi thí sinh có nền tảng toán học và tư duy thuật toán sâu hơn.

1.4.3 Các cuộc thi trực tuyến

Bên cạnh các kỳ thi chính thức, hiện nay có rất nhiều **cuộc thi trực tuyến** mở cho mọi đối tượng tham gia.

Nền tảng tổ chức contest trực tuyến phổ biến nhất hiện nay là **Codeforces**, với các cuộc thi diễn ra gần như hàng tuần. Trên Codeforces, người chơi được chia thành hai hạng:

- Div.2: dành cho người mới và trung cấp
- Div.1: dành cho người có kinh nghiệm

Ngoài Codeforces, còn nhiều nền tảng khác như:

- AtCoder

- HackerRank
- TopCoder
- CS Academy

Một số công ty công nghệ lớn cũng tổ chức các cuộc thi trực tuyến kết hợp vòng chung kết trực tiếp, tiêu biểu như:

- Google Code Jam
- Facebook Hacker Cup
- Yandex.Algorithm

Các cuộc thi này không chỉ mang tính học thuật mà còn được sử dụng như một kênh tuyển dụng, giúp thí sinh thể hiện năng lực giải quyết vấn đề và thuật toán.

1.4.4 Sách và tài liệu tham khảo

Ngoài việc tham gia các kỳ thi, sách và tài liệu chuyên sâu là nguồn học tập rất quan trọng đối với người học lập trình thi đấu.

Sách về lập trình thi đấu

- S. S. Skiena, M. A. Revilla – *Programming Challenges*
- S. Halim, F. Halim – *Competitive Programming 3*
- K. Diks et al. – *Looking for a Challenge?*

Hai cuốn đầu phù hợp với người mới bắt đầu, trong khi cuốn cuối chứa nhiều bài toán nâng cao.

Sách thuật toán tổng quát

- T. H. Cormen et al. – *Introduction to Algorithms*
- J. Kleinberg, É. Tardos – *Algorithm Design*
- S. S. Skiena – *The Algorithm Design Manual*

Các tài liệu này cung cấp nền tảng lý thuyết vững chắc, rất hữu ích cho việc học và nghiên cứu thuật toán chuyên sâu.

CHƯƠNG 2

KIỂU DỮ LIỆU – TOÁN TỬ – NHẬP XUẤT

Người soạn: Đặng Phúc An Khang – Khóa 2 ngành Công nghệ Thông tin
Tài liệu tham khảo chính:

- *Competitive Programmer's Handbook* – Antti Laaksonen
- *C++* – 28tech

Contents

2.1 Kiểu dữ liệu

Trong lập trình thi đấu, việc lựa chọn **kiểu dữ liệu** phù hợp ảnh hưởng trực tiếp đến tính đúng đắn của chương trình. Một lỗi rất thường gặp là **tràn số** (overflow) khi dùng kiểu quá nhỏ, hoặc sai số khi xử lý **số thực**. Phần này trình bày các kiểu dữ liệu cơ bản thường dùng: **số nguyên**, **số thực**, **ký tự** và **đúng/sai**.

2.1.1 Kiểu dữ liệu số nguyên (Integers)

Kiểu số nguyên dùng để lưu các giá trị *không có phần thập phân*. Trong C++, các kiểu số nguyên phổ biến như sau:

Kiểu dữ liệu	Kích thước (byte)	Giá trị có thể lưu	Phạm vi giá trị có thể lưu
`short`	2	Signed integer	$-32{,}768 \to 32{,}767$
`unsigned short`	2	Unsigned integer	$0 \to 65{,}535$
`int`	4	Signed integer	$-2{,}147{,}483{,}648 \to 2{,}147{,}483{,}647$
`unsigned int`	4	Unsigned integer	$0 \to 4{,}294{,}967{,}295$
`long long`	8	Signed integer	$-9{,}223{,}372{,}036{,}854{,}775{,}808 \to 9{,}223{,}372{,}036{,}854{,}775{,}80$
`unsigned long long`	8	Unsigned integer	$0 \to 18{,}446{,}744{,}073{,}709{,}551{,}615$

Cách tính phạm vi theo số bit

Dựa theo số byte mà kiểu dữ liệu cần để lưu trữ, ta có thể tính ra phạm vi giá trị mà kiểu số nguyên đó có thể biểu diễn. Vì 1 byte = 8 bit, nếu một kiểu dữ liệu có K bit thì:

- **Số nguyên có dấu (signed):**
$$-2^{K-1} \ \to \ 2^{K-1} - 1.$$
Ví dụ, `int` thường có 4 byte = 32 bit nên phạm vi là $-2^{31} \to 2^{31} - 1$.
- **Số nguyên không dấu (unsigned):**
$$0 \ \to \ 2^{K} - 1.$$
Ví dụ, `unsigned int` thường có 32 bit nên phạm vi là $0 \to 2^{32} - 1$.

Thông thường, mỗi kiểu số nguyên có dấu sẽ có một kiểu không dấu tương ứng. Kiểu có dấu sử dụng 1 bit để biểu diễn dấu âm/dương, còn kiểu không dấu dùng toàn bộ K bit để biểu diễn giá trị, vì vậy phạm vi không âm của kiểu không dấu thường lớn hơn.

Khi nào dùng `int` và khi nào dùng `long long`?

Trong lập trình thi đấu, kiểu được dùng nhiều nhất là `int` (thường 32-bit) với phạm vi xấp xỉ:

$$-2^{31} \dots 2^{31} - 1 \approx -2 \cdot 10^9 \dots 2 \cdot 10^9.$$

Nếu giá trị có thể vượt ngoài khoảng này (ví dụ tổng lớn, tích lớn, hoặc công thức có thể lên tới 10^{18}), ta nên dùng `long long` (thường 64-bit) với phạm vi xấp xỉ:

$$-2^{63} \dots 2^{63} - 1 \approx -9 \cdot 10^{18} \dots 9 \cdot 10^{18}.$$

Lưu ý quan trọng: overflow do biểu thức vẫn là `int`

Một lỗi rất phổ biến là biến kết quả có kiểu `long long`, nhưng biểu thức trung gian vẫn bị tính bằng `int`. Ví dụ sau chứa lỗi tinh vi:

```
int a = 123456789;
long long b = a * a;
cout << b << "\n"; // wrong because a*a overflows in int
```

Nguyên nhân: cả hai toán hạng của `a*a` đều là `int`, nên phép nhân được thực hiện bằng `int` trước. Cách sửa an toàn là ép kiểu một vế hoặc dùng hằng `1LL`:

```
long long b1 = (long long)a * a; // cast before multiplication
long long b2 = 1LL * a * a;      // force long long multiplication
```

Kiểu 128-bit (tham khảo)

Trình biên dịch `g++` còn hỗ trợ kiểu 128-bit `__int128_t` (phạm vi rất lớn, xấp xỉ $\pm 10^{38}$). Tuy nhiên, kiểu này không phải lúc nào cũng có trên mọi hệ thống chấm, nên chỉ dùng khi thật sự cần.

2.1.2 Số học modulo (Modular arithmetic)

Ký hiệu $x \bmod m$ là số dư khi chia x cho m. Ví dụ:

$$17 \bmod 5 = 2 \quad \text{vì} \quad 17 = 3 \cdot 5 + 2.$$

Trong nhiều bài toán, đáp án có thể rất lớn nhưng đề bài chỉ yêu cầu in theo modulo, ví dụ "mod $10^9 + 7$". Khi đó, ta có thể lấy modulo sau mỗi phép toán để giá trị không bị quá lớn.

Tính chất quan trọng:

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$
$$(a - b) \bmod m = ((a \bmod m) - (b \bmod m)) \bmod m$$
$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

Ví dụ: tính $n!$ modulo m:

```
long long x = 1;
for (int i = 2; i <= n; i++) {
    x = (x * i) % m; // keep numbers small by taking modulo each step
}
cout << x << "\n";
```

Lưu ý về số âm: trong C++, nếu x âm thì $x\%m$ có thể âm. Nếu muốn kết quả luôn trong $[0, m-1]$:

```
x %= m;              // take remainder
if (x < 0) x += m;   // normalize to [0, m-1]
```

2.1.3 Kiểu dữ liệu số thực (Floating point)

Khi cần lưu giá trị có phần thập phân, ta dùng kiểu số thực. Trong C++ có ba kiểu số thực phổ biến: `float`, `double`, `long double`.

Kiểu dữ liệu	Kích thước (byte)	Độ chính xác (tham khảo)
`float`	4	6 – 9 decimal digits
`double`	8	15 – 18 decimal digits
`long double`	16	33 – 36 decimal digits

Thông thường, `double` là lựa chọn đủ tốt cho đa số bài toán. Dù số thực có thể lưu số nguyên, ta không nên dùng số thực để thay thế số nguyên vì số thực có thể có rounding error.

In số thực theo độ chính xác yêu cầu

Độ chính xác cần thiết thường được nêu trong đề bài (ví dụ: in ra 9 chữ số sau dấu phẩy). Với `cout`, ta dùng thư viện `iomanip` để định dạng:

```
#include <iomanip>

double x = 3.141592653589793;
cout << fixed << setprecision(9) << x << "\n"; // print 9 digits after decimal
```

Trong đó:

- `fixed` buộc in theo dạng thập phân cố định.
- `setprecision(9)` đặt số chữ số sau dấu phẩy là 9.

Sai số làm tròn và so sánh số thực

Một số giá trị không thể biểu diễn chính xác tuyệt đối dưới dạng số thực, dẫn đến sai số. Ví dụ:

```
double x = 0.3*3 + 0.1;
cout << fixed << setprecision(20) << x << "\n";
// may print 0.99999999999999988898
```

Vì vậy, không nên so sánh số thực bằng `==`. Thay vào đó, coi hai số bằng nhau nếu sai khác nhỏ hơn một ngưỡng nhỏ ε:

```
double eps = 1e-9;
if (abs(a - b) < eps) {
    // treat a and b as equal
}
```

2.1.4 Kiểu dữ liệu ký tự (Character)

Khi cần lưu trữ một ký tự, ta sử dụng kiểu `char`. Các ký tự như chữ cái, chữ số và ký tự đặc biệt đều có thể lưu bằng `char`. Mỗi ký tự có một mã số (ASCII) tương ứng, vì vậy `char` cũng có thể được xem như một số nguyên nhỏ.

Kiểu dữ liệu	Kích thước (byte)	Giá trị có thể lưu	Phạm vi giá trị có thể lưu
`char`	1	Integer or character	$-128 \to 127$
`unsigned char`	1	Non-negative integer or character	$0 \to 255$

2.1.5 Kiểu đúng/sai (Boolean)

Kiểu đúng sai (luận lý) được dùng để lưu trữ giá trị `true` hoặc `false`. Trong C++, kiểu `bool` được sử dụng cho mục đích này và thường có kích thước khoảng 1 byte.

```
bool ok = true;
if (ok) cout << "YES\n";  // print YES if ok is true
else cout << "NO\n";      // otherwise print NO
```

2.2 Biến và từ khóa

2.2.1 Khai báo biến

Biến (variable) được sử dụng để lưu trữ các giá trị cần thiết trong quá trình giải bài toán. Hầu hết mọi chương trình đều sử dụng biến, ví dụ để lưu chiều dài, chiều rộng, kết quả tính toán, trạng thái đúng/sai, v.v.

Mỗi biến khi khai báo cần xác định rõ:

- **Kiểu dữ liệu** của biến (ví dụ: `int`, `long long`, `double`, `char`, `bool`).
- **Tên biến** do người lập trình đặt.
- **Giá trị khởi tạo** (có thể có hoặc không).

Cú pháp chung:

```
<kiểu dữ liệu> <tên biến>;
```

Hoặc khai báo kèm khởi tạo giá trị:

```
<kiểu dữ liệu> <tên biến> = <giá trị>;
```

Lưu ý: nếu khai báo biến mà không khởi tạo giá trị ban đầu, biến có thể chứa *giá trị rác* (không xác định).

Ví dụ 1: Khai báo và khởi tạo biến

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 28;
    long long b = 282828282828282LL;
    float c = 3.8912f;
    double d = 10.8912781;
    char ch = '@';
    bool ok = true;

    cout << "Value of a: " << a << "\n";
    cout << "Value of b: " << b << "\n";
    cout << "Value of c: " << c << "\n";
    cout << "Value of d: " << d << "\n";
    cout << "Value of ch: " << ch << "\n";
    cout << "Value of ok: " << ok << "\n";

    return 0;
}
```

Kết quả:

```
Value of a: 28
Value of b: 282828282828282
```

```
Value of c: 3.8912
Value of d: 10.8913
Value of ch: @
Value of ok: 1
```

Giải thích: với kiểu `bool`, giá trị `true` được in ra dưới dạng `1`, và `false` được in ra dưới dạng `0`.

Ví dụ 2: Khai báo nhiều biến cùng kiểu trên một dòng

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 10, b = 20, c = 30;
    char x = '@', y = '#', z = '$';

    cout << a << "␣" << b << "␣" << c << "\n";
    cout << x << "␣" << y << "␣" << z << "\n";

    return 0;
}
```

Kết quả:

```
10 20 30
@ # $
```

2.2.2 Quy tắc đặt tên biến

Ngôn ngữ C++ quy định một số quy tắc để tên biến được coi là hợp lệ. Ngoài ra, người lập trình nên đặt tên biến sao cho *ngắn gọn, dễ hiểu và có ý nghĩa.*

Các quy tắc cơ bản:

- Tên biến **không được bắt đầu bằng chữ số**. Ví dụ sai: `21APC`, `300radius`.
- Tên biến **không được chứa dấu cách hoặc ký tự đặc biệt**. Ví dụ sai: `ban kinh`, `ban#kinh`.
- Tên biến **không được trùng với từ khóa của C++**. Ví dụ sai: `int`, `using`, `for`.
- Không được khai báo **hai biến cùng tên** trong cùng một phạm vi. Ví dụ sai: `int a; double a;`
- C++ **phân biệt chữ hoa và chữ thường**. `tech28` và `Tech28` là hai biến khác nhau.

Khuyến nghị khi đặt tên biến:

- Đặt tên có ý nghĩa, phản ánh nội dung biến lưu trữ.
- Tránh đặt tên quá dài hoặc quá khó hiểu.
- Có thể dùng:
  - **CamelCase**: `BanKinh`, `ChuVi`, `DienTich`.
  - **snake_case**: `ban_kinh`, `chu_vi`, `dien_tich`.

2.2.3 Từ khóa (Keywords)

Từ khóa (keywords) là các từ được định nghĩa sẵn trong ngôn ngữ C++ và có ý nghĩa đặc biệt. Người lập trình **không được** sử dụng từ khóa làm tên biến hay tên hàm.

Một số từ khóa thường gặp:

- Kiểu dữ liệu: `int`, `long`, `double`, `char`, `bool`, `void`.
- Điều khiển luồng: `if`, `else`, `for`, `while`, `break`, `continue`.
- Khác: `return`, `using`, `namespace`, `true`, `false`.

Thực tế, người học **không cần học thuộc toàn bộ từ khóa**. Khi học đến kiến thức nào, các từ khóa liên quan sẽ tự nhiên xuất hiện và được ghi nhớ qua quá trình sử dụng.

2.3 Input và Output

Trong lập trình thi đấu, chương trình thường đọc dữ liệu đầu vào (input) và in kết quả ra (output) thông qua các luồng chuẩn. Với ngôn ngữ C++, hai luồng chuẩn được sử dụng phổ biến nhất là `cin` để đọc dữ liệu và `cout` để ghi dữ liệu.

2.3.1 Đọc dữ liệu cơ bản

Dữ liệu đầu vào trong các bài toán lập trình thi đấu thường bao gồm các số và chuỗi ký tự, được phân tách bởi dấu cách hoặc ký tự xuống dòng. Ta có thể đọc các dữ liệu này bằng toán tử `>>` của `cin` như sau:

```
int a, b;
string x;
cin >> a >> b >> x;
```

Câu lệnh trên sẽ lần lượt đọc hai số nguyên `a`, `b` và một chuỗi ký tự `x` từ input. Cách đọc này hoạt động tốt miễn là giữa các dữ liệu có ít nhất một dấu cách hoặc một dòng mới.

Ví dụ, cả hai dạng input sau đều được đọc đúng:

Dạng 1:

```
123 456 monkey
```

Dạng 2:

```
123 456
monkey
```

2.3.2 Ghi dữ liệu ra màn hình

Sau khi xử lý xong dữ liệu, chương trình cần in kết quả ra output. Trong C++, việc này thường được thực hiện bằng `cout`:

```
int a = 123, b = 456;
string x = "monkey";
cout << a << "␣" << b << "␣" << x << "\n";
```

Ký tự `"\n"` dùng để xuống dòng sau khi in xong kết quả.

2.3.3 Tăng tốc Input và Output

Trong các bài toán có lượng dữ liệu lớn, việc đọc và ghi dữ liệu có thể làm chương trình chạy chậm. Để cải thiện tốc độ, người lập trình thường thêm hai dòng lệnh sau vào đầu chương trình:

```
ios::sync_with_stdio(false);
cin.tie(nullptr);
```

Hai dòng lệnh này giúp `cin` và `cout` hoạt động nhanh hơn. Ngoài ra, nên sử dụng `"\n"` thay cho `endl`, vì `endl` luôn thực hiện thao tác làm rỗng bộ đệm, gây chậm chương trình.

2.3.4 Đọc cả dòng dữ liệu

Trong một số bài toán, dữ liệu đầu vào là một dòng văn bản hoàn chỉnh có thể chứa dấu cách. Khi đó, ta sử dụng hàm `getline`:

```
string s;
getline(cin, s);
```

Hàm `getline` sẽ đọc toàn bộ một dòng, bao gồm cả các khoảng trắng, cho đến khi gặp ký tự xuống dòng.

2.3.5 Đọc dữ liệu cho đến khi hết input

Nếu số lượng dữ liệu đầu vào không được cho trước, ta có thể sử dụng vòng lặp sau:

```
while (cin >> x) {
    // code
}
```

Vòng lặp này sẽ liên tục đọc dữ liệu cho đến khi không còn dữ liệu trong input.

2.3.6 Input và Output từ tệp

Trong một số hệ thống thi hoặc khi test chương trình trên máy cá nhân, dữ liệu vào và ra được lưu trong tệp. Ta có thể sử dụng các lệnh sau để đọc và ghi dữ liệu từ tệp:

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

Sau khi sử dụng các lệnh trên, chương trình sẽ đọc dữ liệu từ tệp `input.txt` và ghi kết quả ra tệp `output.txt`, trong khi phần còn lại của code vẫn giữ nguyên.

2.4 Kiểu dữ liệu `pair`

Trong lập trình, đặc biệt là lập trình thi đấu, rất nhiều bài toán yêu cầu lưu trữ dữ liệu theo **cặp giá trị**. Ví dụ:

- Tọa độ điểm (x, y)
- Thời gian bắt đầu và kết thúc của một công việc
- Đỉnh đầu và đỉnh cuối của một cạnh trong đồ thị

C++ cung cấp kiểu dữ liệu `pair` để giải quyết trực tiếp nhu cầu này.

2.4.1 Giới thiệu về `pair`

`pair` là một container nằm trong thư viện `utility`, cho phép lưu trữ **hai giá trị đi kèm nhau** nhưng chỉ sử dụng **một biến**.

Đặc điểm của `pair`

- Phần tử thứ nhất gọi là `first`
- Phần tử thứ hai gọi là `second`
- Có thể gán, sao chép, so sánh như các kiểu dữ liệu thông thường
- Truy cập phần tử bằng toán tử dấu chấm `.`

Cú pháp khai báo

```
// Method 1: Declare a pair with default-initialized values
pair<T1, T2> p;

// Method 2: Initialize using make_pair (explicitly creates a pair from two values)
pair<T1, T2> p = make_pair(value1, value2);

// Method 3: Initialize using the constructor
pair<T1, T2> p(value1, value2);

// Method 4: Initialize using brace-initialization (uniform initialization)
pair<T1, T2> p = {value1, value2};
```

T1 và T2 có thể là: `int`, `long long`, `double`, `char`, `string` hoặc thậm chí là `pair` khác.

2.4.2 Ví dụ cơ bản với pair

Ví dụ 1: Các cách khai báo pair

```
#include <iostream>
#include <utility>
using namespace std;

int main() {
    pair<int, int> a = make_pair(28, 100);
    cout << a.first << "␣" << a.second << "\n";

    pair<char, int> b = {'@', 28};
    cout << b.first << "␣" << b.second << "\n";

    pair<char, char> c('#', '$');
    cout << c.first << "␣" << c.second << "\n";

    return 0;
}
```

Output:

```
28 100
@ 28
# $
```

2.4.3 Pair lồng nhau (Nested pair)

Một `pair` có thể chứa một `pair` khác, giúp biểu diễn cấu trúc dữ liệu phức tạp hơn.

Ví dụ 2: Pair lồng nhau

```
#include <iostream>
#include <utility>
using namespace std;

int main() {
    pair<int, pair<char, int>> p1 = make_pair(28, make_pair('@', 100));
    cout << p1.first << "\n";
    cout << p1.second.first << "␣" << p1.second.second << "\n";

    pair<pair<int, int>, pair<int, int>> p2 = {{10, 20}, {30, 40}};
    cout << p2.first.first << "␣" << p2.first.second << "\n";
    cout << p2.second.first << "␣" << p2.second.second << "\n";

    return 0;
}
```

Output:

```
28
@ 100
10 20
30 40
```

2.4.4 Toán tử với `pair`

`pair` hỗ trợ các toán tử:

- Gán (`=`)
- So sánh (`==`, `!=`, `<`, `>`, `<=`, `>=`)
- Hoán đổi (`swap`)

Toán tử gán

Khi gán một `pair` cho `pair` khác, giá trị của `first` và `second` sẽ được sao chép.

```cpp
#include <iostream>
#include <utility>
using namespace std;

int main() {
    pair<string, int> p = make_pair("APC", 21);
    pair<string, int> p1 = p;
    cout << p1.first << "␣" << p1.second << "\n";
    return 0;
}
```

Output:

```
APC 21
```

So sánh `pair`

Khi so sánh hai `pair`:

- So sánh `first` trước
- Nếu bằng nhau thì so sánh `second`

```cpp
#include <iostream>
#include <utility>
using namespace std;

int main() {
    pair<int, int> p1 = {10, 20};
    pair<int, int> p2 = {10, 21};
    pair<int, int> p3 = {5, 35};

    cout << boolalpha << (p1 == p2) << "\n";
    cout << boolalpha << (p1 != p2) << "\n";
    cout << boolalpha << (p1 < p2) << "\n";
    cout << boolalpha << (p1 > p3) << "\n";

    return 0;
}
```

Output:

```
false
true
true
true
```

Hoán đổi hai `pair`

```cpp
#include <iostream>
#include <utility>
using namespace std;

int main() {
    pair<int, int> p1 = {10, 20};
    pair<int, int> p2 = {30, 40};

    cout << "Before swap:\n";
    cout << "p1 = {" << p1.first << ", " << p1.second << "}\n";
    cout << "p2 = {" << p2.first << ", " << p2.second << "}\n";

    p1.swap(p2);

    cout << "After swap:\n";
    cout << "p1 = {" << p1.first << ", " << p1.second << "}\n";
    cout << "p2 = {" << p2.first << ", " << p2.second << "}\n";

    return 0;
}
```

Output:

```
Before swap:
p1 = {10, 20}
p2 = {30, 40}
After swap:
p1 = {30, 40}
p2 = {10, 20}
```

2.5 Chú thích (Comment)

Chú thích (comment) là một phần rất quan trọng khi viết chương trình. Chú thích được dùng để ghi chú, giải thích và làm rõ ý tưởng của mã nguồn, giúp người đọc (hoặc chính người viết trong tương lai) dễ dàng hiểu được chương trình đang làm gì.

Chú thích **không được coi là mã nguồn thực thi** và sẽ bị trình biên dịch bỏ qua hoàn toàn khi biên dịch chương trình.

2.5.1 Vì sao cần chú thích khi viết code

Chú thích mang lại nhiều lợi ích và là một thói quen tốt của lập trình viên:

- **Giải thích mã nguồn:** Chú thích có thể được sử dụng như mã giả (pseudo-code) để mô tả ý tưởng và chức năng của từng đoạn chương trình. Khi đọc lại code cũ hoặc khi người khác đọc code, họ có thể nhanh chóng nắm được logic tổng thể.
- **Mô tả thuật toán:** Khi thuật toán trở nên phức tạp, việc chú thích giúp giải thích rõ cách hoạt động, các bước xử lý và ý tưởng chính của thuật toán.
- **Metadata (siêu dữ liệu):** Chú thích có thể dùng để mô tả thông tin về file mã nguồn như: tên chương trình, tác giả, mục đích, ngày viết, hoặc các lưu ý quan trọng.

- **Hỗ trợ gỡ lỗi (debugging):** Trong quá trình debug, ta có thể tạm thời vô hiệu hóa một hoặc nhiều dòng code bằng cách chuyển chúng thành chú thích thay vì xóa hẳn.

Nhìn chung, việc sử dụng chú thích hợp lý giúp code **dễ đọc, dễ hiểu và dễ bảo trì hơn**.

2.5.2 Chú thích trên một dòng

Trong C++, chú thích trên một dòng được tạo bằng hai dấu gạch chéo `//`. Mọi nội dung nằm sau `//` trên cùng một dòng sẽ được coi là chú thích.

Ví dụ:

```cpp
#include <bits/stdc++.h>
using namespace std;

// This is a single-line comment

int main() {
    // Declare three integer variables and initialize them
    int a = 10, b = 20, c = 30;

    // Declare three character variables and initialize them
    char x = '@', y = '#', z = '$';

    // Print values of a, b, and c
    cout << a << "␣" << b << "␣" << c << "\n";

    // Print values of x, y, and z
    cout << x << "␣" << y << "␣" << z << "\n";

    return 0;
}
```

Kết quả:

```
10 20 30
@ # $
```

Ghi chú: Trong nhiều trình soạn thảo code, bạn có thể dùng tổ hợp phím `Ctrl + /` để nhanh chóng chú thích hoặc bỏ chú thích nhiều dòng code cùng lúc, rất hữu ích khi debug.

2.5.3 Chú thích trên nhiều dòng

Để chú thích trên nhiều dòng, ta đặt nội dung cần chú thích giữa `/*` và `*/`. Cách này thường dùng khi cần mô tả dài hoặc chú thích một khối mã lớn.

Ví dụ:

```cpp
#include <bits/stdc++.h>
using namespace std;

/*
 This is a multi-line comment.
 It can span multiple lines.
 Useful for describing programs or algorithms.
*/

int main() {
    /*
     This program declares three integer variables
     and prints their values to the screen.
    */
    int a = 10, b = 20, c = 30;
    cout << a << "␣" << b << "␣" << c << "\n";
    return 0;
}
```

Kết quả:

```
10 20 30
```

2.5.4 Lưu ý khi sử dụng chú thích

- Chú thích nên **ngắn gọn, rõ ràng** và đúng trọng tâm.
- Tránh chú thích những điều quá hiển nhiên (ví dụ: `i++ // increase i`).
- Nên chú thích **ý tưởng và mục đích**, không chỉ mô tả lại từng dòng code.
- Trong lập trình thi đấu, ưu tiên chú thích cho:
  - Ý tưởng thuật toán.
  - Các đoạn xử lý quan trọng.
  - Các trường hợp đặc biệt (edge cases).

2.6 Toán tử (Operator)

Toán tử (operator) là các ký hiệu đặc biệt được dùng để thực hiện phép tính, so sánh, kiểm tra điều kiện và thao tác trên dữ liệu. Trong C++, toán tử xuất hiện trong hầu hết mọi chương trình, vì vậy việc hiểu rõ cách hoạt động của chúng là điều bắt buộc.

2.6.1 Phân loại toán tử

Các toán tử thường dùng trong C++ có thể chia thành các nhóm chính sau:

- Toán tử gán
- Toán tử toán học
- Toán tử so sánh
- Toán tử logic
- Toán tử tăng giảm
- Toán tử điều kiện (toán tử ba ngôi)

2.6.2 Toán tử gán

Toán tử gán (`=`) dùng để gán giá trị của một biểu thức cho một biến.

Cú pháp:

$$X = Y$$

Ý nghĩa: giá trị của biểu thức Y được tính trước, sau đó gán cho biến X.

Ví dụ: gán và gán lại giá trị

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = 28;             // assign 28 to n
    cout << "Value␣of␣n:␣" << n << "\n";

    int m = n;              // m gets value of n
```

```
    cout << "Value of m: " << m << "\n";

    int p = m;              // p gets value of m
    p = 56;                 // reassign p
    cout << "Value of p: " << p << "\n";

    return 0;
}
```

Output:

```
Value of n: 28
Value of m: 28
Value of p: 56
```

Lưu ý: phép gán khác hoàn toàn phép so sánh. `=` khác với `==`.

2.6.3 Toán tử toán học

Các toán tử toán học cơ bản trong C++ gồm:

Toán tử	Ý nghĩa	Ví dụ
`+`	Phép cộng	`a + b`
`-`	Phép trừ	`a - b`
`*`	Phép nhân	`a * b`
`/`	Phép chia	`a / b`
`%`	Phép chia dư	`a % b`

Thứ tự ưu tiên toán tử

Trong một biểu thức:

- `*`, `/`, `%` có ưu tiên cao hơn `+`, `-`
- dấu ngoặc `()` có ưu tiên cao nhất
- cùng mức ưu tiên thì tính từ trái sang phải

Ví dụ: độ ưu tiên và dấu ngoặc

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int x = 2 + 3 * 4;      // 2 + (3*4)
    int y = (2 + 3) * 4;    // (2+3) * 4
    cout << x << "\n";
    cout << y << "\n";
    return 0;
}
```

Output:

```
14
20
```

Ví dụ 1: chu vi và diện tích hình chữ nhật

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int length, width;
    cin >> length >> width;

    int perimeter = 2 * (length + width);
    int area = length * width;

    cout << "Perimeter: " << perimeter << "\n";
    cout << "Area: " << area << "\n";
    return 0;
}
```

Sample input:

```
5 3
```

Output:

```
Perimeter: 16
Area: 15
```

Ví dụ 2: chu vi và diện tích hình tròn

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    double r;
    cin >> r;

    double perimeter = 2 * 3.14 * r;
    double area = 3.14 * r * r;

    cout << fixed << setprecision(2);
    cout << "Perimeter: " << perimeter << "\n";
    cout << "Area: " << area << "\n";
    return 0;
}
```

Sample input:

```
2
```

Output:

```
Perimeter: 12.56
Area: 12.56
```

Chú ý 1: chia nguyên và chia thực

Nếu cả hai toán hạng đều là số nguyên, phép chia là **chia nguyên** (mất phần thập phân):

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 10, b = 3;
    cout << a / b << "\n";   // integer division
    return 0;
}
```

Output:

```
3
```

Muốn có phần thập phân, cần ép kiểu hoặc nhân với `1.0`:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 10, b = 3;
    cout << fixed << setprecision(2);
    cout << (double)a / b << "\n"; // casting
    cout << 1.0 * a / b << "\n";   // multiply by 1.0
    return 0;
}
```

Output:

```
3.33
3.33
```

Chú ý 2: tràn số khi nhân

Một lỗi phổ biến trong lập trình thi đấu là tràn số khi nhân hai số `int` lớn:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 1000000, b = 1000000;

    long long wrong = a * b;          // overflow happens here
    long long correct1 = (long long)a * b;
    long long correct2 = 1LL * a * b;

    cout << wrong << "\n";
    cout << correct1 << "\n";
    cout << correct2 << "\n";
    return 0;
}
```

Output (thường gặp trên máy 32-bit int):

```
-727379968
1000000000000
1000000000000
```

Giải thích: dù biến `wrong` là `long long`, nhưng biểu thức `a*b` được tính bằng `int` trước, nên đã tràn rồi mới gán sang `long long`.

2.6.4 Toán tử gán kết hợp

Các toán tử gán kết hợp giúp viết code ngắn gọn hơn:

Toán tử	Tương đương
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`
`a %= b`	`a = a % b`

Ví dụ: gán kết hợp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 10;
    a += 5;   // a = 15
    a *= 2;   // a = 30
    a -= 7;   // a = 23
    cout << a << "\n";
    return 0;
}
```

Output:

```
23
```

2.6.5 Toán tử so sánh

Toán tử so sánh dùng để so sánh hai giá trị và trả về `true` hoặc `false`.

Toán tử	Ý nghĩa
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Khác

Ví dụ: in ra kết quả so sánh

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    cout << boolalpha; // print true/false instead of 1/0

    cout << (20 > 10) << "\n";
    cout << (30 < 50) << "\n";
    cout << (10 != 10) << "\n";
    cout << (10 == 10) << "\n";

    return 0;
}
```

Output:

```
true
true
false
true
```

2.6.6 Toán tử logic

Ba toán tử logic cơ bản:

- `&&` (AND): đúng nếu tất cả đều đúng
- `||` (OR): đúng nếu có ít nhất một đúng
- `!` (NOT): phủ định

Ví dụ: AND/OR/NOT

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    bool res1 = (10 < 20) && (20 >= 20);      // true && true
    bool res2 = (10 > 20) || (5 == 5);        // false || true
    bool res3 = !(10 < 20);                   // !true

    cout << res1 << "␣" << res2 << "␣" << res3 << "\n";
    return 0;
}
```

Output:

```
1 1 0
```

Lưu ý: khi in `bool` mặc định sẽ ra `1/0`. Nếu muốn in `true/false`, dùng `boolalpha`.

2.6.7 Toán tử tăng giảm

Toán tử tăng (`++`) và giảm (`-`) (2 dấu - liền kề nhau) thay đổi giá trị biến thêm hoặc bớt 1 đơn vị.

- `++x`: tăng trước (pre-increment)
- `x++`: tăng sau (post-increment)
- `-x`: giảm trước (pre-decrement)
- `x-`: giảm sau (post-decrement)

Ví dụ: tăng/giảm cơ bản

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = 100;

    ++n;  // n = 101
    cout << n << "\n";

    n++;  // n = 102
    cout << n << "\n";

    n--;  // n = 101
    cout << n << "\n";

    --n;  // n = 100
    cout << n << "\n";

    return 0;
}
```

Output:

```
101
102
101
100
```

Ví dụ: khác nhau giữa n++ và ++n

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = 100;

    cout << n++ << "\n"; // prints 100, then n becomes 101
    cout << n << "\n";   // prints 101

    cout << ++n << "\n"; // n becomes 102, then prints 102
    cout << n << "\n";   // prints 102

    return 0;
}
```

Output:

```
100
101
102
102
```

2.6.8 Toán tử điều kiện (Toán tử ba ngôi)

Toán tử ba ngôi là dạng rút gọn của `if-else`.

Cú pháp:

```
condition ? value_if_true : value_if_false
```

Ví dụ: toán tử ba ngôi

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = (10 < 20) ? 28 : 82;
    cout << n << "\n";

    n = ((50 < 50) && (10 > 3)) ? 28 : 82;
    cout << n << "\n";

    return 0;
}
```

Output:

```
28
82
```

2.7 Rút gọn code

Trong lập trình thi đấu, thời gian viết code là yếu tố rất quan trọng. Vì vậy, các lập trình viên thường rút gọn code bằng cách đặt tên ngắn hơn cho kiểu dữ liệu, cấu trúc dữ liệu và các thao tác thường dùng.

2.7.1 Rút gọn tên kiểu dữ liệu (typedef / using)

Trong C++, ta có thể đặt tên mới (ngắn hơn) cho một kiểu dữ liệu đã tồn tại. Điều này giúp code gọn gàng và dễ viết hơn.

Sử dụng `typedef`

Ví dụ, kiểu `long long` rất hay dùng nhưng tên khá dài. Ta có thể đặt tên rút gọn là `ll`:

```
typedef long long ll;
```

Sau đó, hai đoạn code sau là hoàn toàn tương đương:

```
long long a = 123456789;
long long b = 987654321;
cout << a * b << "\n";
```

```
ll a = 123456789;
ll b = 987654321;
cout << a * b << "\n";
```

Output:

```
121932631112635269
```

Rút gọn các kiểu phức tạp

`typedef` đặc biệt hữu ích với các kiểu dữ liệu dài như `vector` hay `pair`:

```
typedef vector<int> vi;
typedef pair<int,int> pi;
```

Ví dụ sử dụng:

```
vi v;
v.push_back(10);
v.push_back(20);

pi p = make_pair(3, 5);
cout << p.first << "␣" << p.second << "\n";
```

Output:

```
3 5
```

Cách viết hiện đại: `using`

Trong C++ hiện đại, có thể dùng `using` thay cho `typedef`:

```
using ll = long long;
using vi = vector<int>;
using pi = pair<int,int>;
```

Hai cách này là tương đương, nhưng `using` dễ đọc hơn với kiểu phức tạp.

2.7.2 Macros (`#define`)

Macro cho phép thay thế một đoạn văn bản trước khi chương trình được biên dịch. Trong C++, macro được định nghĩa bằng từ khóa `#define`.

Macro rút gọn thao tác phổ biến

Ví dụ các macro thường gặp trong lập trình thi đấu:

```
#define F first
#define S second
#define PB push_back
#define MP make_pair
```

Khi đó, đoạn code sau:

```
vector<pair<int,int>> v;
v.push_back(make_pair(y1, x1));
v.push_back(make_pair(y2, x2));
int d = v[i].first + v[i].second;
```

có thể được rút gọn thành:

```
vector<pair<int,int>> v;
v.PB(MP(y1, x1));
v.PB(MP(y2, x2));
int d = v[i].F + v[i].S;
```

Ưu điểm:

- Code ngắn hơn
- Viết nhanh hơn khi thi

Nhược điểm:

- Khó đọc với người mới
- Dễ gây lỗi nếu lạm dụng

2.7.3 Macro có tham số

Macro cũng có thể nhận tham số, thường dùng để rút gọn vòng lặp.

Ví dụ:

```
#define REP(i,a,b) for (int i = a; i <= b; i++)
```

Sau đó, vòng lặp:

```
for (int i = 1; i <= n; i++) {
    search(i);
}
```

có thể viết gọn lại:

```
REP(i,1,n) {
    search(i);
}
```

Lưu ý: macro không kiểm tra cú pháp như hàm, nên cần dùng cẩn thận.

2.7.4 Lỗi thường gặp khi dùng macro

Xét macro tính bình phương:

```
#define SQ(a) a*a
```

Macro này có thể gây lỗi nghiêm trọng.

Ví dụ:

```
cout << SQ(3 + 3) << "\n";
```

Sau khi thay thế macro, code thực tế trở thành:

```
cout << 3 + 3 * 3 + 3 << "\n";
```

Output:

```
15
```

Kết quả sai so với mong đợi là 36.

Cách viết macro an toàn hơn

Luôn đặt tham số trong dấu ngoặc:

```
#define SQ(a) ((a) * (a))
```

Khi đó:

```
cout << SQ(3 + 3) << "\n";
```

Output:

```
36
```

2.8 Thư viện `cmath` và `algorithm`

Trong quá trình lập trình, đặc biệt là lập trình thi đấu, việc sử dụng các hàm toán học và các hàm thuật toán có sẵn giúp chương trình ngắn gọn, rõ ràng và ít lỗi hơn. Ngôn ngữ C++ cung cấp hai thư viện rất quan trọng:

- `cmath` (hoặc `math.h`): chứa các hàm toán học
- `algorithm`: chứa các hàm thuật toán cơ bản

2.8.1 Các hàm toán học phổ biến (`cmath`)

Hầu hết các hàm trong thư viện `cmath`:

- Nhận tham số kiểu `double`
- Trả về kết quả kiểu `double`

STT	Hàm	Chức năng
1	`pow(x,y)`	Tính x^y
2	`sqrt(x)`	Căn bậc hai của x
3	`cbrt(x)`	Căn bậc ba của x
4	`ceil(x)`	Làm tròn lên
5	`floor(x)`	Làm tròn xuống
6	`round(x)`	Làm tròn gần nhất
7	`fabs(x)`	Giá trị tuyệt đối
8	`exp(x)`	e^x
9	`fmod(x,y)`	Phần dư của x/y
10	`log(x)`	Logarit tự nhiên
11	`log10(x)`	Logarit cơ số 10
12	`sin(x)`	Hàm sin (radian)
13	`cos(x)`	Hàm cos (radian)
14	`tan(x)`	Hàm tan (radian)

Hàm `pow`

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

int main() {
```

```
    int a = 2, b = 10;
    cout << a << "^" << b << " = " << (int)pow(a, b) << "\n";

    int n = 100;
    double root = pow(n, 1.0 / 5);
    cout << fixed << setprecision(3) << root << "\n";
}
```

Output:

```
2^10 = 1024
2.512
```

Lưu ý: `pow` trả về `double`, cần ép kiểu nếu muốn số nguyên.

Hàm `sqrt`

```
int n = 100;
cout << (int)sqrt(n) << "\n";

double x = sqrt(1000);
cout << fixed << setprecision(2) << x << "\n";
```

Output:

```
10
31.62
```

Hàm `ceil`, `floor`, `round`

```
double a = 3.14, b = 3.8;
cout << (int)ceil(a) << " " << (int)ceil(b) << "\n";
cout << (int)floor(a) << " " << (int)floor(b) << "\n";
cout << (int)round(a) << " " << (int)round(b) << "\n";
```

Output:

```
4 4
3 3
3 4
```

Hàm trị tuyệt đối

```
cout << abs(10) << " " << abs(-100) << "\n";
```

Output:

```
10 100
```

2.8.2 Thư viện `algorithm`

Thư viện `algorithm` cung cấp nhiều hàm xử lý dữ liệu rất hữu ích. Ở giai đoạn này, ta làm quen với một số hàm cơ bản.

Hàm `swap`

```cpp
#include <algorithm>

int a = 100, b = 200;
swap(a, b);
cout << a << "␣" << b << "\n";
```

Output:

```
200 100
```

Hàm `min`

```cpp
cout << min(100, 200) << "\n";
cout << min({10, 20, 15, 4}) << "\n";
cout << min({'d', 'b', 'z'}) << "\n";
```

Output:

```
100
4
b
```

Hàm `max`

```cpp
cout << max(100, 200) << "\n";
cout << max({10, 20, 15, 4}) << "\n";
cout << max({'d', 'b', 'z'}) << "\n";
```

Output:

```
200
20
z
```

CHƯƠNG 3

CẤU TRÚC RẼ NHÁNH

Contents

3.1 Cấu trúc rẽ nhánh `if - else` trong C++

Trong lập trình, rất nhiều bài toán yêu cầu chương trình phải **ra quyết định**: nếu điều kiện đúng thì làm A, nếu sai thì làm B. Trong C++, cấu trúc `if - else` là công cụ cơ bản để xử lý những tình huống này.

3.1.1 Biểu thức điều kiện là gì?

- **Biểu thức điều kiện** là một biểu thức có giá trị `true` (đúng) hoặc `false` (sai).
- Thường là các phép so sánh: `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Có thể kết hợp nhiều điều kiện bằng toán tử logic: `&&` (AND), `||` (OR), `!` (NOT).

Trong C++:

- `0` được xem là **false**.
- Mọi giá trị **khác 0** được xem là **true**.

Ví dụ, các điều kiện sau đều có kiểu `bool`:

```cpp
// Check if x equals 10
(x == 10)

// Check if n is even
(n % 2 == 0)

// Check if 20 <= n <= 50
(n >= 20 && n <= 50)

// Check if n is divisible by 3 or 5
(n % 3 == 0 || n % 5 == 0)
```

3.1.2 Câu lệnh `if`

Câu lệnh `if` cho phép bạn **chỉ thực hiện một khối lệnh nếu điều kiện đúng**.

3.1.2.1 Cú pháp cơ bản

```cpp
if (condition) {
    // code block executed when condition is true
}
```

Ý nghĩa:

- Nếu `condition` đúng (`true`) → khối lệnh bên trong `{}` được chạy.
- Nếu `condition` sai (`false`) → khối lệnh bị bỏ qua.

3.1.2.2 Ví dụ 1: Kiểm tra giá trị cụ thể

Yêu cầu: Nếu `n == 28` thì in ra `APC`.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 28;

    // If n equals 28, print "APC"
    if (n == 28) {
        cout << "APC\n";
    }

    // This condition is false, so this line will not be printed
    if (n > 30) {
        cout << "APC Blog\n";
    }

    cout << "end\n";
    return 0;
}
```

Output:

```
APC
end
```

3.1.2.3 Ví dụ 2: Kiểm tra số chẵn

Một số nguyên **là số chẵn** nếu chia cho 2 dư 0.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 28;

    // If n is even, print "CHAN"
    if (n % 2 == 0) {
        cout << "CHAN\n";
    }

    // If n is odd, print "LE"
    if (n % 2 != 0) {
        cout << "LE\n";
    }

    return 0;
}
```

Output:

```
CHAN
```

3.1.2.4 Ví dụ 3: Kiểm tra tính chia hết

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 28, m = 4;

    // Check if n is divisible by m
    if (n % m == 0) {
        cout << "Chia␣het\n";
    }

    if (n % m != 0) {
        cout << "Khong␣chia␣het\n";
    }

    return 0;
}
```

Output:

```
Chia het
```

3.1.2.5 Ví dụ 4: Kết hợp nhiều điều kiện

Yêu cầu: Nếu `n` thuộc một trong các số {2, 3, 5, 7} thì in `YES`.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 5;

    // Check if n is one of 2, 3, 5 or 7
    if (n == 2 || n == 3 || n == 5 || n == 7) {
        cout << "YES\n";
```

```
    }

    return 0;
}
```

Output:

```
YES
```

3.1.2.6 Lưu ý: dùng số làm điều kiện

Vì mọi số khác 0 được xem là `true`, ta có thể viết:

```
#include <iostream>
using namespace std;

int main() {
    int n = 28;
    int m = 0;

    // n is non-zero -> condition is true
    if (n) {
        cout << "n is non-zero\n";
    }

    // m is zero -> condition is false
    if (m) {
        cout << "m is non-zero\n";
    }

    cout << "END\n";
    return 0;
}
```

Output:

```
n is non-zero
END
```

3.1.3 Câu lệnh `if - else`

Câu lệnh `if - else` cho phép ta xử lý **cả hai nhánh**:

- Khi điều kiện đúng
- Khi điều kiện sai

3.1.3.1 Cú pháp

```
if (condition) {
    // executed when condition is true
} else {
    // executed when condition is false
}
```

Lưu ý:

- `else` luôn đi kèm với một `if` trước đó.
- Chỉ **một** trong hai khối `if` hoặc `else` được thực thi.

3.1.3.2 Ví dụ 1: Kiểm tra chẵn / lẻ (có `else`)

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;

    // If n is even, print "CHAN" and "APC"
    if (n % 2 == 0) {
        cout << "CHAN\n";
        cout << "APC\n";
    } else {
        // Otherwise, print "LE" and the APC website
        cout << "LE\n";
        cout << "sot.umtoj.edu.vn\n";
    }

    return 0;
}
```

Ví dụ input: 28

Output:

```
CHAN
APC
```

3.1.3.3 Ví dụ 2: Kiểm tra năm nhuận

Định nghĩa năm nhuận:

- Chia hết cho 400, **hoặc**
- Chia hết cho 4 nhưng **không** chia hết cho 100.

```cpp
#include <iostream>
using namespace std;

int main() {
    int year;
    cin >> year;

    bool isLeap =
        (year % 400 == 0) ||
        (year % 4 == 0 && year % 100 != 0);

    if (isLeap) {
        cout << "YES\n";
    } else {
        cout << "NO\n";
    }

    return 0;
}
```

Input:

```
2020
```

Output:

```
YES
```

3.1.4 Chuỗi `if - else if - else`

Khi có nhiều trường hợp cần phân loại, ta có thể sử dụng **nhiều điều kiện nối tiếp nhau**:

```
if (condition1) {
    // case 1
} else if (condition2) {
    // case 2
} else if (condition3) {
    // case 3
} else {
    // default case
}
```

Cách hoạt động:

- Các điều kiện được kiểm tra **từ trên xuống dưới**.
- Ngay khi gặp điều kiện đúng, khối lệnh tương ứng được thực thi,
- Các điều kiện phía sau **không được kiểm tra nữa**.

3.1.4.1 Ví dụ: Phân loại điểm

```
#include <iostream>
using namespace std;

int main() {
    int score;
    cin >> score;

    if (score >= 90) {
        cout << "Excellent\n";
    } else if (score >= 75) {
        cout << "Good\n";
    } else if (score >= 50) {
        cout << "Average\n";
    } else {
        cout << "Need improvement\n";
    }

    return 0;
}
```

3.1.5 `if - else` lồng nhau

Khi điều kiện phức tạp, ta có thể **chia nhỏ** bằng cách đặt `if - else` bên trong một `if - else` khác.

3.1.5.1 Ví dụ: Kiểm tra đoạn và chia hết

Yêu cầu:

- $20 \leq n \leq 50$
- và n chia hết cho ít nhất một trong các số $2, 3, 5, 7$

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
```

```
7
8       // First, check if n is in the range [20, 50]
9       if (n >= 20 && n <= 50) {
10
11          // Then, check divisibility by 2, 3, 5 or 7
12          if (n % 2 == 0 || n % 3 == 0 ||
13              n % 5 == 0 || n % 7 == 0) {
14              cout << "YES\n";
15          } else {
16              cout << "NO\n";
17          }
18
19      } else {
20          cout << "NO\n";
21      }
22
23      return 0;
24  }
```

3.1.6 Một số lưu ý quan trọng

- Luôn dùng dấu ngoặc nhọn `{}` cho khối lệnh, kể cả khi chỉ có một dòng, để tránh bug khi thêm dòng mới sau này.
- Cẩn thận với toán tử gán `=` và so sánh `==`.
- Nên tách điều kiện phức tạp thành nhiều biến trung gian `bool` để code dễ đọc hơn.
- Khi nhiều nhánh **loại trừ lẫn nhau**, ưu tiên dùng `if - else if - else` thay vì nhiều `if` rời rạc.

Ví dụ: sử dụng biến `bool` giúp code dễ đọc hơn

```
1   bool inRange = (n >= 20 && n <= 50);
2   bool divisible = (n % 2 == 0 || n % 3 == 0 ||
3                     n % 5 == 0 || n % 7 == 0);
4
5   if (inRange && divisible) {
6       cout << "YES\n";
7   } else {
8       cout << "NO\n";
9   }
```

3.2 `if` và `else if` (nhiều nhánh điều kiện)

Trong nhiều bài toán, chương trình không chỉ có hai trường hợp "đúng/sai" mà có thể có **nhiều nhánh** khác nhau (mỗi nhánh ứng với một điều kiện). Nếu chỉ dùng `if - else` thì bạn thường phải lồng nhiều lớp `if`, khiến code dài và khó đọc. Khi đó, cấu trúc `if - else if - else` giúp viết chương trình gọn hơn và rõ ràng hơn.

3.2.1 Cú pháp và cách hoạt động

```
1   // The program checks conditions from top to bottom.
2   // It executes the first block whose condition is true, then stops checking.
3   if (condition_1) {
4       // block 1
5   }
6   else if (condition_2) {
7       // block 2
8   }
9   else if (condition_3) {
10      // block 3
11  }
```

```
else {
    // default block (optional)
}
```

Ghi nhớ nhanh:

- Có thể có **bao nhiêu** nhánh `else if` cũng được.
- Nhánh `else` **có thể có hoặc không**.
- Hệ thống kiểm tra điều kiện **từ trên xuống dưới**.
- Chỉ chạy **duy nhất 1 nhánh đầu tiên đúng**, sau đó **kết thúc** cấu trúc rẽ nhánh.

3.2.2 Ví dụ 1: Xếp loại điểm bằng `else if`

```
#include <iostream>
using namespace std;

int main() {
    int score;
    cin >> score;

    // Classify the score into categories
    if (score >= 90) {
        cout << "Excellent\n";
    }
    else if (score >= 75) {
        cout << "Good\n";
    }
    else if (score >= 50) {
        cout << "Average\n";
    }
    else {
        cout << "Need␣improvement\n";
    }

    return 0;
}
```

Input:

```
76
```

Output:

```
Good
```

3.2.3 Bài tập áp dụng

3.2.3.1 Bài 1: Số ngày của tháng

Yêu cầu: Nhập tháng m và năm y, in ra số ngày của tháng đó. Chú ý: tháng 2 có 29 ngày nếu năm nhuận, 28 ngày nếu không nhuận.

Quy tắc năm nhuận:

- Năm chia hết cho 400 $\Rightarrow$ năm nhuận, hoặc
- Năm chia hết cho 4 và không chia hết cho 100 $\Rightarrow$ năm nhuận.

```cpp
#include <iostream>
using namespace std;

int main() {
    int m, y;
    cin >> m >> y;

    // Months with 31 days
    if (m == 1 || m == 3 || m == 5 || m == 7 ||
        m == 8 || m == 10 || m == 12) {
        cout << 31 << "\n";
    }
    // Months with 30 days
    else if (m == 4 || m == 6 || m == 9 || m == 11) {
        cout << 30 << "\n";
    }
    // February
    else if (m == 2) {
        bool isLeap = (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);

        if (isLeap) cout << 29 << "\n";
        else cout << 28 << "\n";
    }
    // Invalid month
    else {
        cout << "Invalid␣month\n";
    }

    return 0;
}
```

Ví dụ 1:

Input:

```
2 2020
```

Output:

```
29
```

Ví dụ 2:

Input:

```
2 2021
```

Output:

```
28
```

Ví dụ 3:

Input:

```
13 2024
```

Output:

```
Invalid month
```

3.2.3.2 Bài 2: Phân loại tam giác

Yêu cầu: Nhập 3 cạnh nguyên a, b, c (giả sử luôn tạo thành tam giác hợp lệ). In:

- 1 nếu tam giác đều
- 2 nếu tam giác cân
- 3 nếu tam giác vuông
- 4 nếu tam giác thường

Ghi chú quan trọng:

- Kiểm tra "đều" trước vì "đều" cũng là một dạng của "cân".
- Kiểm tra "vuông" dùng định lý Pythagoras: $a^2 = b^2 + c^2$ hoặc $b^2 = a^2 + c^2$ hoặc $c^2 = a^2 + b^2$.

```cpp
#include <iostream>
using namespace std;

int main() {
    long long a, b, c;
    cin >> a >> b >> c;

    // Equilateral triangle
    if (a == b && b == c) {
        cout << 1 << "\n";
    }
    // Isosceles triangle
    else if (a == b || b == c || a == c) {
        cout << 2 << "\n";
    }
    // Right triangle (use long long to avoid overflow in squaring)
    else if (a * a == b * b + c * c ||
             b * b == a * a + c * c ||
             c * c == a * a + b * b) {
        cout << 3 << "\n";
    }
    // Scalene triangle
    else {
        cout << 4 << "\n";
    }

    return 0;
}
```

Ví dụ 1:

Input:

```
3 3 3
```

Output:

```
1
```

Ví dụ 2:

Input:

```
5 5 8
```

Output:

```
2
```

Ví dụ 3:

Input:

```
3 4 5
```

Output:

```
3
```

Ví dụ 4:

Input:

```
4 5 6
```

Output:

```
4
```

3.3 Bảng mã ASCII và thư viện `cctype`

3.3.1 Bảng mã ASCII là gì?

ASCII (American Standard Code for Information Interchange) là bảng mã gán **mỗi ký tự** (chữ cái, chữ số, ký hiệu, ...) với một **số nguyên** tương ứng.

Trong thực tế học C++ cơ bản, bạn chỉ cần nhớ một số cụm quan trọng để làm bài nhanh.

Các cụm ASCII nên nhớ

Cụm ký tự	Mã ASCII (thập phân)
'a' - 'z'	97 – 122
'A' - 'Z'	65 – 90
'0' - '9'	48 – 57

Lưu ý quan trọng: Trong C++, kiểu `char` lưu ký tự, nhưng bản chất nó vẫn là một số nhỏ (thường 1 byte). Khi bạn dùng `char` trong biểu thức số học, chương trình sẽ dùng **mã ASCII** của ký tự đó.

3.3.2 Ví dụ: `char` hoạt động như số (ASCII)

Ví dụ 1: In mã ASCII của ký tự

```cpp
#include <bits/stdc++.h>
using namespace std;

```

```cpp
int main() {
    char c1 = 'A', c2 = 'a', c3 = '0';

    // Print ASCII codes by casting to int
    cout << (int)c1 << "␣" << (int)c2 << "␣" << (int)c3 << "\n";

    int n = c1; // n becomes 65
    cout << n << "\n";

    // 'A' = 65, so 'A' + 100 = 165
    cout << (c1 + 100) << "\n";
    return 0;
}
```

Output:

```
65 97 48
65
165
```

Ví dụ 2: Tăng/giảm ký tự

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'A';

    // 'A' = 65, so 65 + 10 = 75
    int n = c + 10;
    cout << n << "\n";

    // Pre-increment: 'A' -> 'B'
    ++c;
    cout << (int)c << "\n";
    cout << c << "\n";
    return 0;
}
```

Output:

```
75
66
B
```

3.3.3 Tự kiểm tra loại ký tự (không cần thư viện)

Trong nhiều bài về chuỗi, bạn hay gặp các yêu cầu: *"ký tự này là chữ thường/chữ hoa/chữ số?"* Bạn có thể tự kiểm tra bằng cách so sánh theo khoảng ASCII.

Kiểm tra chữ thường 'a'..'z'

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'u';

    // Method 1: compare by characters
    if (c >= 'a' && c <= 'z') cout << "YES\n";
    else cout << "NO\n";

```

```
    // Method 2: compare by ASCII codes
    if (c >= 97 && c <= 122) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

Kiểm tra chữ hoa 'A'..'Z'

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'u';

    if (c >= 'A' && c <= 'Z') cout << "YES\n";
    else cout << "NO\n";

    if (c >= 65 && c <= 90) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

Kiểm tra chữ cái (hoa hoặc thường)

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'Z';

    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

Kiểm tra chữ số '0'..'9'

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = '5';

    // Method 1
    if (c >= '0' && c <= '9') cout << "YES\n";
    else cout << "NO\n";

    // Method 2
    if (c >= 48 && c <= 57) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

Đổi chữ hoa ↔ chữ thường bằng ASCII

Trong ASCII, chữ thường có mã **lớn hơn** chữ hoa tương ứng **32 đơn vị**:

$$'a' - 'A' = 97 - 65 = 32$$

- Hoa → thường: cộng 32
- Thường → hoa: trừ 32

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'A';
    c += 32; // 'A' -> 'a'
    cout << c << "\n";

    char d = 'z';
    d -= 32; // 'z' -> 'Z'
    cout << d << "\n";

    return 0;
}
```

Output:

```
a
Z
```

3.3.4 Thư viện `cctype`

C++ có sẵn các hàm để kiểm tra/chuyển đổi ký tự trong thư viện `<cctype>` (`<ctype.h>` cũng dùng được).

Hàm	Ý nghĩa
`islower(c)`	1 nếu `c` là chữ thường, ngược lại 0
`isupper(c)`	1 nếu `c` là chữ hoa, ngược lại 0
`isalpha(c)`	1 nếu `c` là chữ cái, ngược lại 0
`isdigit(c)`	1 nếu `c` là chữ số, ngược lại 0
`isalnum(c)`	1 nếu `c` là chữ cái hoặc chữ số, ngược lại 0
`tolower(c)`	trả về ký tự thường tương ứng (nếu có)
`toupper(c)`	trả về ký tự hoa tương ứng (nếu có)

Lưu ý nhỏ khi dùng `cctype`: Các hàm như `isupper`, `tolower` nhận `int` (thực chất là giá trị ký tự), nên trong C++ an toàn nhất là ép `unsigned char` trước khi gọi nếu dữ liệu có thể ngoài ASCII. Với newbie làm bài cơ bản ASCII thì cứ dùng trực tiếp cũng ổn.

Ví dụ 1: Kiểm tra chữ hoa/chữ thường

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'A', d = 'z';

    if (isupper(c)) cout << "YES\n";
    else cout << "NO\n";

    if (islower(d)) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

Output:

```
YES
YES
```

Ví dụ 2: Chuyển đổi hoa/thường

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    char c = 'A', d = 'z';

    c = tolower(c); // 'A' -> 'a'
    d = toupper(d); // 'z' -> 'Z'

    cout << c << "␣" << d << "\n";
    return 0;
}
```

Output:

```
a Z
```

3.4 Bài tập

Bài tập 1. Phân loại nấm **link:** https://marisaoj.com/problem/396

Marisa đang phân tích một cây nấm có chỉ số độc tố `T` (từ `0.0` đến `10.0`).

- Rất độc nếu `T` từ `9.0` trở lên.
- Độc nếu `T` từ `5.0` đến `8.9`.
- An toàn nếu `T` dưới `5.0`.

Input

Một dòng gồm một số thực `T` (có đúng 1 chữ số thập phân).

Output

- In `VERY TOXIC` nếu nấm rất độc.
- In `TOXIC` nếu nấm độc.
- In `SAFE` nếu nấm an toàn.

Ví dụ

Dữ liệu vào	Kết quả ra
5.0	TOXIC

Hướng giải.

- Đọc số thực `T`.
- So sánh theo ngưỡng:
  - Nếu `T >= 9.0` → `VERY TOXIC`.
  - Ngược lại nếu `T >= 5.0` → `TOXIC`.
  - Còn lại → `SAFE`.

- Vì dữ liệu có đúng 1 chữ số thập phân nên so sánh trực tiếp kiểu `double` là đủ an toàn.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    double T;
    cin >> T;

    if(T >= 9.0){
        cout << "VERY TOXIC";
    }else if(T >= 5.0){
        cout << "TOXIC";
    }else{
        cout << "SAFE";
    }

    return 0;
}
```

Bài tập 2. Phương trình nghiệm nguyên **link:** https://marisaoj.com/problem/397

Tìm nghiệm nguyên của phương trình:

$$ax + b = 0.$$

Trong đó **nghiệm nguyên** nghĩa là x phải là một số nguyên.

Input

Một dòng gồm hai số nguyên `a, b`.

Output

- Nếu có đúng một nghiệm nguyên, in ra nghiệm đó.
- Nếu có vô số nghiệm, in ra `INFINITE SOLUTIONS`.
- Nếu không tồn tại nghiệm nguyên, in ra `NO SOLUTION`.

Điều kiện

$$-100 \leq a, b \leq 100$$

Ví dụ

Dữ liệu vào	Kết quả ra
3 6	-2

Hướng giải.

- Xét các trường hợp của hệ số `a`:
  - Nếu $a = 0$:
    - Nếu $b = 0$ thì phương trình $0x + 0 = 0$ đúng với mọi $x \Rightarrow$ vô số nghiệm.
    - Nếu $b \neq 0$ thì $0x + b = 0$ vô lý $\Rightarrow$ không có nghiệm.
  - Nếu $a \neq 0$:
    - Ta có $x = -\dfrac{b}{a}$.
    - Để x là số nguyên thì b phải chia hết cho a, tức là $b \bmod a = 0$.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b;
    cin >> a >> b;

    if(a == 0){
        if(b == 0) cout << "INFINITE␣SOLUTIONS";
        else cout << "NO␣SOLUTION";
    }else{
        if(b % a == 0){
            cout << (-b / a);
        }else{
            cout << "NO␣SOLUTION";
        }
    }

    return 0;
}
```

Bài tập 3. Tuổi **link:** https://marisaoj.com/problem/420

Có hai người:

- Người thứ nhất sinh ngày `a`, tháng `b`, năm `c`.
- Người thứ hai sinh ngày `x`, tháng `y`, năm `z`.

Biết rằng hai người không sinh cùng ngày. Hãy xác định người nào nhiều tuổi hơn.

Input

Một dòng gồm sáu số nguyên `a, b, c, x, y, z`.

Output

In ra `1` nếu người thứ nhất nhiều tuổi hơn, ngược lại in ra `2`.

Điều kiện

Đảm bảo ngày hợp lệ và hai người không sinh cùng ngày.

Ví dụ

Dữ liệu vào	Kết quả ra
7 10 1990 8 10 1990	1

Hướng giải.

- Người nhiều tuổi hơn là người có ngày sinh **sớm hơn**.
- So sánh theo thứ tự:
  - So sánh năm: năm nhỏ hơn ⇒ sinh sớm hơn.
  - Nếu năm bằng nhau, so sánh tháng: tháng nhỏ hơn ⇒ sinh sớm hơn.
  - Nếu tháng bằng nhau, so sánh ngày: ngày nhỏ hơn ⇒ sinh sớm hơn.
- Nếu ngày sinh của người thứ nhất sớm hơn người thứ hai ⇒ in `1`, ngược lại in `2`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c, x, y, z;
    cin >> a >> b >> c >> x >> y >> z;
```

```
8
9       // so sanh nam truoc
10      if(c != z){
11          if(c < z) cout << 1;
12          else cout << 2;
13          return 0;
14      }
15
16      // neu nam bang nhau, so sanh thang
17      if(b != y){
18          if(b < y) cout << 1;
19          else cout << 2;
20          return 0;
21      }
22
23      // neu thang bang nhau, so sanh ngay
24      if(a < x) cout << 1;
25      else cout << 2;
26
27      return 0;
28  }
```

Bài tập 4. Ghép hình chữ nhật **link:** https://marisaoj.com/problem/421

Cho hai hình chữ nhật có kích thước lần lượt `a×b` và `c×d`. Kiểm tra xem có thể ghép hai hình chữ nhật này thành một hình chữ nhật khác bằng cách đặt chúng cạnh nhau hay không.

Input

Một dòng gồm bốn số nguyên `a, b, c, d`.

Output

In ra `YES` nếu có thể ghép, ngược lại in ra `NO`.

Điều kiện

$$1 \leq a, b, c, d \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
1 2 3 4	NO

Hướng giải.

- Ghép "cạnh nhau" có 2 kiểu:
  - Ghép ngang: hai hình phải có **cùng chiều cao**.
  - Ghép dọc: hai hình phải có **cùng chiều rộng**.
- Mỗi hình có thể xoay, tức là (a, b) hoặc (b, a), và (c, d) hoặc (d, c).
- Ta xét 4 cách xoay, với mỗi cách kiểm tra:
  - $H_1 = H_2$ (ghép ngang) hoặc $W_1 = W_2$ (ghép dọc).

Cài đặt

```
1   #include <bits/stdc++.h>
2   #define int long long
3   using namespace std;
4
5   signed main(){
6       int a, b, c, d;
7       cin >> a >> b >> c >> d;
8
9       int ok = 0;
10
11      // Case 1: (a,b) va (c,d)
```

```
    if(b == d) ok = 1; // ghep ngang (cao bang nhau)
    if(a == c) ok = 1; // ghep doc (rong bang nhau)

    // Case 2: (a,b) va (d,c)
    if(b == c) ok = 1;
    if(a == d) ok = 1;

    // Case 3: (b,a) va (c,d)
    if(a == d) ok = 1;
    if(b == c) ok = 1;

    // Case 4: (b,a) va (d,c)
    if(a == c) ok = 1;
    if(b == d) ok = 1;

    cout << (ok ? "YES" : "NO");
    return 0;
}
```

Bài tập 5. Tam giác **link:** https://marisaoj.com/problem/4

Nhập vào ba số nguyên `a, b, c`. Hãy kiểm tra xem có thể tạo được một tam giác từ ba cạnh có độ dài `a, b, c` hay không.

Input

Một dòng gồm 3 số nguyên `a, b, c`.

Output

In ra `YES` nếu có thể tạo thành tam giác, ngược lại in ra `NO`.

Điều kiện

$$1 \leq a, b, c \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
3 4 5	YES

Hướng giải.

- Ba đoạn thẳng tạo thành tam giác khi và chỉ khi **tổng độ dài hai cạnh bất kỳ lớn hơn cạnh còn lại**.
- Tức là cần đồng thời thỏa:

$$a + b > c, \quad a + c > b, \quad b + c > a.$$

- Nếu cả 3 bất đẳng thức đều đúng thì in `YES`, ngược lại in `NO`.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c;
    cin >> a >> b >> c;

    if(a + b > c && a + c > b && b + c > a){
        cout << "YES";
    }else{
        cout << "NO";
    }

    return 0;
}
```

Bài tập 6. Máy tính **link:** https://marisaoj.com/problem/535

Thiết kế một chiếc máy tính xử lí được bốn phép toán cơ bản: cộng (`+`), trừ (`-`), nhân (`*`) và chia (`/`) với hai số thực `a`, `b`.

Input

Một dòng gồm lần lượt:

- Số thực `a`
- Một kí tự biểu diễn phép toán
- Số thực `b`

Output

- In ra kết quả của phép tính, làm tròn tới `3` chữ số thập phân.
- Nếu phép tính không thể thực hiện được (chia cho 0), in ra `ze`.

Điều kiện

$$-1000 \leq a, b \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
4 * 5	20.000

Hướng giải.

- Đọc hai số thực `a`, `b` và kí tự phép toán.
- Xét từng trường hợp của phép toán:
  - `+`: tính $a + b$
  - `-`: tính $a - b$
  - `*`: tính $a \times b$
  - `/`: nếu $b = 0$ thì không thực hiện được
- Nếu phép toán hợp lệ, in kết quả với định dạng `fixed` và `setprecision(3)`.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    double a, b;
    char op;
    cin >> a >> op >> b;

    double res;
    bool ok = true;

    if(op == '+'){
        res = a + b;
    }else if(op == '-'){
        res = a - b;
    }else if(op == '*'){
        res = a * b;
    }else if(op == '/'){
        if(b == 0){
            ok = false;
        }else{
            res = a / b;
```

```
        }
    }else{
        ok = false;
    }

    if(!ok){
        cout << "ze";
    }else{
        cout << fixed << setprecision(3) << res;
    }

    return 0;
}
```

Bài tập 7. Nhỏ nhất và lớn nhất **link:** https://marisaoj.com/problem/5

Cho ba số nguyên `a, b, c`. Hãy in ra số nhỏ nhất và số lớn nhất trong ba số đó.

Input

Một dòng gồm 3 số nguyên `a, b, c`.

Output

In ra hai số: số nhỏ nhất và số lớn nhất.

Giới hạn

$$1 \leq a, b, c \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
4 3 4	3 4

Hướng giải.

- Khởi tạo `mn` và `mx` lần lượt bằng `a`.
- So sánh `b` và `c` với `mn` để cập nhật giá trị nhỏ nhất.
- So sánh `b` và `c` với `mx` để cập nhật giá trị lớn nhất.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c;
    cin >> a >> b >> c;

    int mn = a, mx = a;

    if(b < mn) mn = b;
    if(c < mn) mn = c;

    if(b > mx) mx = b;
    if(c > mx) mx = c;

    cout << mn << "␣" << mx;
    return 0;
}
```

Bài tập 8. Hình chữ nhật **link:** https://marisaoj.com/problem/7

Cho độ dài ba cạnh `a, b, c` của một hình chữ nhật. Biết rằng trong ba cạnh này có hai cạnh bằng nhau (là hai cạnh đối diện của hình chữ nhật). Hãy in ra độ dài của cạnh còn lại.

Input

Một dòng gồm ba số nguyên `a, b, c`.

Output

In ra độ dài cạnh còn lại của hình chữ nhật.

Điều kiện

$$1 \leq a, b, c \leq 1000$$

Không có trường hợp không hợp lệ.

Ví dụ

Dữ liệu vào	Kết quả ra
4 3 4	3

Hướng giải.

- Trong hình chữ nhật, hai cạnh đối diện có độ dài bằng nhau.
- Do đó trong ba số `a, b, c`:
  - Nếu `a = b` thì cạnh còn lại là `c`.
  - Nếu `a = c` thì cạnh còn lại là `b`.
  - Ngược lại, `b = c` và cạnh còn lại là `a`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c;
    cin >> a >> b >> c;

    if(a == b){
        cout << c;
    }else if(a == c){
        cout << b;
    }else{
        cout << a;
    }

    return 0;
}
```

Bài tập 9. Đoạn thẳng **link:** https://marisaoj.com/problem/419

Cho hai đoạn thẳng `AB` và `CD` nằm trên trục số. Đoạn `AB` kéo dài từ `a` đến `b`, đoạn `CD` kéo dài từ `c` đến `d`. Hãy kiểm tra xem hai đoạn thẳng có điểm chung không.

Input

Một dòng gồm bốn số nguyên `a, b, c, d`.

Output

Nếu hai đoạn thẳng có điểm chung, in ra `YES`. Ngược lại in ra `NO`.

Điều kiện

$$1 \leq a \leq b \leq 1000, \quad 1 \leq c \leq d \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
1 2 2 3	YES

Hướng giải.

- Hai đoạn thẳng $[a, b]$ và $[c, d]$ có điểm chung khi chúng **giao nhau**.
- Chúng **không** có điểm chung chỉ khi một đoạn nằm hoàn toàn bên trái đoạn kia:

$$b < c \quad \text{hoặc} \quad d < a.$$

- Nếu không rơi vào hai trường hợp trên thì chắc chắn có ít nhất một điểm chung ⇒ in `YES`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    if(b < c || d < a){
        cout << "NO";
    }else{
        cout << "YES";
    }

    return 0;
}
```

Bài tập 10. Tăng dần **link:** https://marisaoj.com/problem/6

Cho ba số nguyên `a, b, c`. Hãy in ra ba số theo thứ tự từ bé đến lớn.

Input

Một dòng gồm 3 số nguyên `a, b, c`.

Output

In ra 3 số theo thứ tự tăng dần.

Giới hạn

$$1 \leq a, b, c \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
8 3 9	3 8 9

Hướng giải.

- Ta sắp xếp ba số bằng cách so sánh và đổi chỗ.
- Nếu `a > b` thì đổi chỗ `a` và `b`.
- Nếu `a > c` thì đổi chỗ `a` và `c`.
- Nếu `b > c` thì đổi chỗ `b` và `c`.
- Sau các bước trên, ta có `a` $\leq$ `b` $\leq$ `c`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c;
    cin >> a >> b >> c;

```

```
    if(a > b){
        int t = a; a = b; b = t;
    }
    if(a > c){
        int t = a; a = c; c = t;
    }
    if(b > c){
        int t = b; b = c; c = t;
    }

    cout << a << "␣" << b << "␣" << c;
    return 0;
}
```

Bài tập 11. Số chính phương **link:** https://marisaoj.com/problem/11

Cho số nguyên a. Hãy xác định xem a có phải là số chính phương hay không.

Input

Một số nguyên a.

Output

In ra YES nếu a là số chính phương, ngược lại in ra NO.

Điều kiện

$$1 \leq a \leq 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
16	YES

Hướng giải.

- Một số là số chính phương nếu tồn tại số nguyên x sao cho $x^2 = a$.
- Ta lấy căn bậc hai của a, làm tròn xuống để được x.
- Nếu $x \times x = a$ thì a là số chính phương.
- Cách này chạy rất nhanh và phù hợp với a lớn tới 10^{18}.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a;
    cin >> a;

    int x = sqrt((long double)a);
    if(x * x == a){
        cout << "YES";
    }else{
        cout << "NO";
    }

    return 0;
}
```

Bài tập 12. Chữ hoa chữ thường **link:** https://marisaoj.com/problem/13

Cho một chữ cái c. Nếu c là chữ thường, hãy in ra c dưới dạng chữ hoa, và ngược lại.

Input

Một chữ cái c (chữ cái trong bảng chữ cái tiếng Anh).

Output

Nếu `c` là chữ hoa, in ra `c` dưới dạng chữ thường; nếu `c` là chữ thường, in ra `c` dưới dạng chữ hoa.

Ví dụ

Dữ liệu vào	Kết quả ra
C	c

Hướng giải.

- Trong bảng mã ASCII:
  - Chữ hoa nằm trong đoạn từ `'A'` đến `'Z'`.
  - Chữ thường nằm trong đoạn từ `'a'` đến `'z'`.
- Nếu `c` là chữ hoa, ta đổi sang chữ thường bằng cách cộng hiệu `'a' - 'A'`.
- Nếu `c` là chữ thường, ta đổi sang chữ hoa bằng cách trừ hiệu `'a' - 'A'`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    char c;
    cin >> c;

    if(c >= 'A' && c <= 'Z'){
        // doi sang chu thuong
        c = c + ('a' - 'A');
    }else{
        // doi sang chu hoa
        c = c - ('a' - 'A');
    }

    cout << c;
    return 0;
}
```

Bài tập 13. Đếm chữ **link:** https://marisaoj.com/problem/14

Cho hai chữ cái `a`, `b` (có thể là chữ hoa hoặc chữ thường). Hãy đếm số lượng chữ cái **nằm giữa** `a` và `b` trong bảng chữ cái tiếng Anh.

Input

Một dòng gồm 2 chữ cái `a, b`.

Output

In ra số lượng chữ cái giữa `a` và `b` trong bảng chữ cái tiếng Anh.

Ví dụ

Dữ liệu vào	Kết quả ra
a E	3

Hướng giải.

- Chuyển cả hai chữ cái về cùng một dạng (ví dụ: chữ thường) để dễ so sánh.
- Khi đã cùng dạng, vị trí của chữ cái trong bảng chữ cái có thể tính bằng:

$$pos = c -' a'$$

(với `a` có vị trí 0, `b` có vị trí 1, ...).

- Số chữ cái nằm giữa hai chữ là:
$$|pos(a) - pos(b)| - 1.$$
- Ví dụ: `a` và `e` có khoảng cách 4, nên ở giữa có $4 - 1 = 3$ chữ: `b, c, d`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    char a, b;
    cin >> a >> b;

    // chuyen ve chu thuong neu la chu hoa
    if(a >= 'A' && a <= 'Z') a = a + ('a' - 'A');
    if(b >= 'A' && b <= 'Z') b = b + ('a' - 'A');

    int posa = a - 'a';
    int posb = b - 'a';

    int ans = llabs(posa - posb) - 1;
    cout << ans;

    return 0;
}
```

Bài tập 14. Định dạng thời gian **link:** https://marisaoj.com/problem/416

Hãy đổi từ `d` giây sang dạng giờ, phút, giây.

Input

Một dòng gồm số nguyên `d` (số giây).

Output

In ra ba số nguyên `h, m, s` lần lượt là số giờ, phút và giây tương ứng.

Điều kiện

$$1 \le d \le 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
3929	1 5 29

Hướng giải.

- Một giờ có 3600 giây, một phút có 60 giây.
- Số giờ:
$$h = \left\lfloor \frac{d}{3600} \right\rfloor$$
- Số giây còn lại sau khi lấy giờ:
$$d = d \bmod 3600$$
- Số phút:
$$m = \left\lfloor \frac{d}{60} \right\rfloor$$
- Số giây còn lại chính là:
$$s = d \bmod 60$$

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int d;
    cin >> d;

    int h = d / 3600;
    d %= 3600;

    int m = d / 60;
    int s = d % 60;

    cout << h << "␣" << m << "␣" << s;
    return 0;
}
```

Bài tập 15. Hóa đơn tiền điện **link:** https://marisaoj.com/problem/587

Tiền điện hàng tháng được tính theo các mức sau:

- Từ kWh `0` đến `50`: mỗi kWh giá `a` đồng.
- Từ kWh `51` đến `100`: mỗi kWh giá `b` đồng.
- Từ kWh `101` đến `150`: mỗi kWh giá `c` đồng.
- Từ kWh `151` trở đi: mỗi kWh giá `d` đồng.

Cho số kWh sử dụng là `x` và bốn số nguyên `a, b, c, d`. Hãy tính tổng số tiền điện phải trả.

Input

Một dòng gồm năm số nguyên `x, a, b, c, d`.

Output

In ra một số nguyên là số tiền điện.

Điều kiện

$$1 \leq x, a, b, c, d \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
60 1 2 3 4	70

Hướng giải.

- Tiền điện được tính theo từng bậc, không phải tất cả kWh đều tính cùng một giá.
- Ta lần lượt trừ số kWh đã tính ở các mức trước:
  - Nếu `x` lớn hơn 50, tính 50 kWh đầu với giá `a`.
  - Nếu còn vượt 100, tính tiếp 50 kWh với giá `b`.
  - Nếu còn vượt 150, tính tiếp 50 kWh với giá `c`.
  - Phần còn lại (nếu có) tính với giá `d`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int x, a, b, c, d;
```

```
    cin >> x >> a >> b >> c >> d;

    int cost = 0;

    if(x <= 50){
        cost = x * a;
    }else if(x <= 100){
        cost = 50 * a + (x - 50) * b;
    }else if(x <= 150){
        cost = 50 * a + 50 * b + (x - 100) * c;
    }else{
        cost = 50 * a + 50 * b + 50 * c + (x - 150) * d;
    }

    cout << cost;
    return 0;
}
```

Bài tập 16. Ăn nấm **link:** https://marisaoj.com/problem/588

Vào những ngày **không phải cuối tuần** (từ thứ Hai đến thứ Sáu), Marisa sẽ ăn một cây nấm. Marisa bắt đầu ăn nấm từ `thứ x`, hỏi sau `y` ngày Marisa đã ăn bao nhiêu nấm.

Input

Một dòng gồm hai số nguyên `x, y`.

Output

In ra số nấm Marisa đã ăn.

Điều kiện

$$2 \le x \le 8 \text{ (8 là Chủ Nhật)}, \qquad 1 \le y \le 10^6$$

Ví dụ

Dữ liệu vào	Kết quả ra
6 5	3

Hướng giải.

- Quy ước: `2..8` tương ứng Thứ Hai..Chủ Nhật.
- Gọi $\texttt{full} = \lfloor y/7 \rfloor$ là số tuần đầy đủ, khi đó ăn được $\texttt{full} \times 5$ nấm.
- Phần còn lại $\texttt{rem} = y \bmod 7$ chỉ nằm trong 7 ngày đầu của một tuần, có thể tính bằng công thức:
  - Đặt $s = x - 2$ (đổi về chỉ số 0..6), với 0..4 là ngày ăn, 5..6 là cuối tuần.
  - Trong `rem` ngày, ta xét các chỉ số ngày $s, s+1, \ldots, s+\texttt{rem}-1$ theo modulo 7.
  - Số ngày ăn bằng `rem`− (số ngày rơi vào cuối tuần).
- Số ngày cuối tuần trong đoạn dài `rem` được tính bằng cách đếm số phần tử rơi vào tập $\{5, 6\}$.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int x, y;
    cin >> x >> y;

    int full = y / 7;
    int rem  = y % 7;

    int ans = full * 5;

```

```
    // s: 0..6 (0=Mon,1=Tue,...,4=Fri,5=Sat,6=Sun)
    int s = x - 2;

    // dem so ngay cuoi tuan trong doan rem ngay
    int weekend = 0;
    if(rem > 0){
        // neu doan rem khong vuot qua het tuan (s+rem-1 <= 6)
        if(s + rem - 1 <= 6){
            // giao voi [5,6]
            int L = s;
            int R = s + rem - 1;
            int l2 = max(L, 5LL);
            int r2 = min(R, 6LL);
            if(l2 <= r2) weekend = r2 - l2 + 1;
        }else{
            // doan bi "vong" qua tuan moi: tach thanh 2 doan
            // Doan 1: [s, 6], Doan 2: [0, (s+rem-1)%7]
            int L1 = s, R1 = 6;
            int l2 = max(L1, 5LL);
            int r2 = min(R1, 6LL);
            if(l2 <= r2) weekend += r2 - l2 + 1;

            int end2 = (s + rem - 1) % 7;
            int L2 = 0, R2 = end2;
            l2 = max(L2, 5LL);
            r2 = min(R2, 6LL);
            if(l2 <= r2) weekend += r2 - l2 + 1;
        }
    }

    ans += (rem - weekend);
    cout << ans;
    return 0;
}
```

CHƯƠNG 4

VÒNG LẶP

Contents

4.1 Vòng lặp `for`

Vòng lặp là một cấu trúc điều khiển cơ bản trong lập trình, cho phép thực hiện một khối lệnh nhiều lần. Trong C++, vòng lặp `for` thường được sử dụng khi số lần lặp đã biết trước hoặc có thể kiểm soát được thông qua một biến đếm.

4.1.1 Cú pháp vòng lặp `for`

```
for (initialization; condition; update) {
    // statements
}
```

Trong đó:

- **initialization**: Câu lệnh khởi tạo, được thực hiện đúng một lần khi bắt đầu vòng lặp (thường dùng để khởi tạo biến đếm).
- **condition**: Điều kiện lặp, được kiểm tra trước mỗi vòng lặp. Nếu điều kiện đúng, khối lệnh bên trong vòng lặp sẽ được thực hiện.
- **update**: Câu lệnh cập nhật, được thực hiện sau mỗi vòng lặp (thường là tăng hoặc giảm biến đếm).

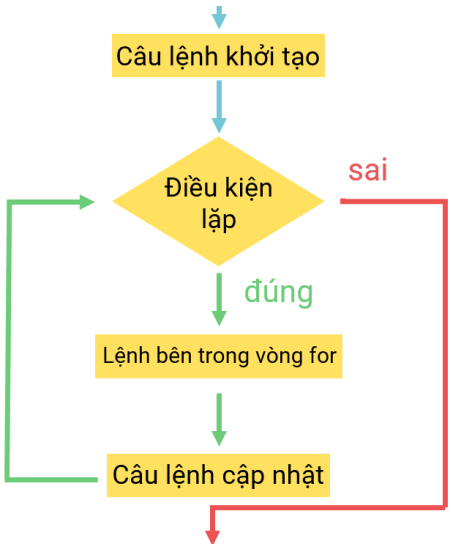
4.1.2 Cơ chế hoạt động

Vòng lặp `for` hoạt động theo trình tự sau:

1. Thực hiện câu lệnh khởi tạo.
2. Kiểm tra điều kiện lặp.
3. Nếu điều kiện đúng, thực hiện khối lệnh trong vòng lặp.
4. Thực hiện câu lệnh cập nhật.
5. Quay lại bước kiểm tra điều kiện.

Khi điều kiện lặp trở thành sai, vòng lặp kết thúc.

4.1.3 Sơ đồ khối vòng lặp `for`



4.1.4 Ví dụ cơ bản

Ví dụ 1: In ra một chuỗi nhiều lần

```cpp
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 4; i++) {
        cout << "APC␣-␣sot.umtoj.edu.vn\n";
    }
    return 0;
}
```

Output:

```
APC - sot.umtoj.edu.vn
APC - sot.umtoj.edu.vn
APC - sot.umtoj.edu.vn
APC - sot.umtoj.edu.vn
```

Ví dụ 2: In các số từ 1 đến n

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 10;
    for (int i = 1; i <= n; i++) {
        cout << i << "␣";
    }
    return 0;
}
```

4.1.5 Ứng dụng thường gặp của vòng lặp `for`

Tính tổng

```cpp
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i;
}
```

Tính giai thừa

```cpp
long long fact = 1;
for (int i = 1; i <= n; i++) {
    fact *= i;
}
```

Duyệt các ước của một số

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

4.1.6 Lệnh `break`

Lệnh `break` được sử dụng để **thoát khỏi vòng lặp ngay lập tức**, bỏ qua tất cả các lần lặp còn lại.

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) break;
    cout << i << "␣";
}
```

Output:

```
1 2 3 4
```

4.1.7 Lệnh `continue`

Lệnh `continue` dùng để **bỏ qua phần còn lại của vòng lặp hiện tại** và chuyển sang lần lặp tiếp theo.

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) continue;
    cout << i << "␣";
}
```

Output:

```
1 2 3 4 6 7 8 9 10
```

4.1.8 Vòng lặp `for` vô hạn

Nếu bỏ điều kiện lặp, vòng lặp sẽ chạy vô hạn:

```
for (;;) {
    // infinite loop
}
```

Trong thực tế, vòng lặp vô hạn thường kết hợp với `break` để chủ động dừng vòng lặp.

4.1.9 Ví dụ: Dừng vòng lặp theo điều kiện

```
#include <iostream>
using namespace std;

int main() {
    int n;
    for (;;) {
        cin >> n;
        if (n == 28) break;
    }
    cout << "Finished\n";
    return 0;
}
```

4.2 Vòng lặp `while`

Bên cạnh vòng lặp `for`, C++ còn cung cấp vòng lặp `while`. Vòng lặp `while` thường được sử dụng khi số lần lặp **chưa biết trước**, và việc lặp phụ thuộc vào một điều kiện được kiểm tra liên tục.

4.2.1 Cú pháp vòng lặp `while`

```
while (condition) {
    // statements
}
```

Trong đó:

- **condition**: Biểu thức điều kiện, được kiểm tra trước mỗi lần lặp. Nếu điều kiện đúng, khối lệnh bên trong vòng lặp được thực hiện.

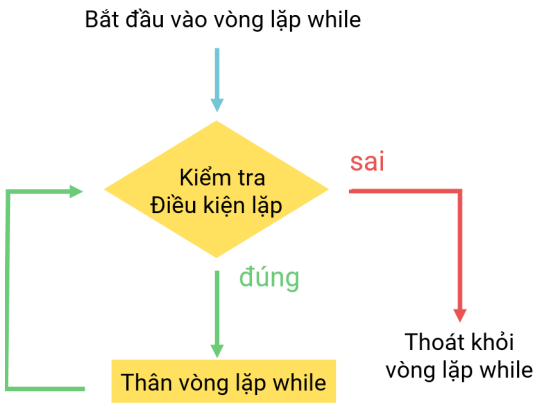
4.2.2 Cơ chế hoạt động

Vòng lặp `while` hoạt động theo trình tự sau:

1. Kiểm tra điều kiện `condition`.
2. Nếu điều kiện sai, vòng lặp kết thúc ngay lập tức.
3. Nếu điều kiện đúng, thực hiện khối lệnh bên trong vòng lặp.
4. Sau khi thực hiện xong, quay lại bước kiểm tra điều kiện.

Nếu điều kiện luôn đúng và không có cơ chế dừng, vòng lặp sẽ trở thành **vòng lặp vô hạn**.

4.2.3 Sơ đồ khối vòng lặp `while`



4.2.4 Ví dụ cơ bản

Ví dụ 1: In các số từ 1 đến n

```
#include <iostream>
using namespace std;

int main() {
    int n = 4;
    int i = 1;
    while (i <= n) {
        cout << i << "␣";
        i++;
    }
    return 0;
}
```

Output:

```
1 2 3 4
```

Giải thích

Ban đầu $i = 1$. Mỗi lần lặp:

- Kiểm tra điều kiện `i <= n`.
- In giá trị của `i`.
- Tăng `i` lên 1.

  Khi $i = 5$, điều kiện sai và vòng lặp kết thúc.

4.2.5 So sánh `while` và `for`

- `for`: dùng khi số lần lặp đã biết trước.
- `while`: dùng khi số lần lặp phụ thuộc vào điều kiện động.

4.2.6 Các bài toán điển hình với vòng lặp `while`

Bài toán 1: Đếm số chữ số của số nguyên dương

Ý tưởng: Mỗi lần loại bỏ chữ số hàng đơn vị bằng phép chia nguyên cho 10.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 12345;
    int count = 0;
    while (n != 0) {
        count++;
        n /= 10;
    }
    cout << count << endl;
    return 0;
}
```

Output:

```
5
```

Bài toán 2: Tính tổng các chữ số của số nguyên

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 12345;
    int sum = 0;
    while (n != 0) {
        sum += n % 10;
        n /= 10;
    }
    cout << sum << endl;
    return 0;
}
```

Output:

```
15
```

4.2.7 Lệnh `break` trong vòng lặp `while`

Lệnh `break` dùng để **thoát khỏi vòng lặp ngay lập tức**, bỏ qua mọi lần lặp còn lại.

```
while (true) {
    int x;
    cin >> x;
    if (x == 28) break;
}
```

Khi nhập giá trị 28, vòng lặp dừng ngay.

4.2.8 Lệnh `continue` trong vòng lặp `while`

Lệnh `continue` bỏ qua phần còn lại của vòng lặp hiện tại và chuyển sang lần lặp tiếp theo.

```
int i = 0;
while (i < 10) {
    i++;
    if (i % 2 == 0) continue;
    cout << i << "␣";
}
```

Output:

```
1 3 5 7 9
```

4.2.9 Vòng lặp `while` vô hạn

```
while (true) {
    // infinite loop
}
```

Trong thực tế, vòng lặp vô hạn **luôn phải đi kèm điều kiện dừng** thông qua `break`.

4.2.10 Ví dụ: Dừng vòng lặp theo điều kiện

```
#include <iostream>
using namespace std;

int main() {
    int n;
    while (true) {
        cin >> n;
        if (n == 28) break;
        cout << "Try␣again\n";
    }
    cout << "Finished\n";
    return 0;
}
```

4.3 Vòng lặp `do-while`

Vòng lặp `do-while` là một biến thể của `while`. Điểm khác biệt quan trọng nhất là: `do-while` **luôn chạy ít nhất một lần**, vì khối lệnh được thực hiện trước rồi mới kiểm tra điều kiện.

4.3.1 Cú pháp vòng lặp `do-while`

```
do {
    // statements
} while (condition);
```

Lưu ý: Sau `while(condition)` bắt buộc có dấu chấm phẩy `;`.

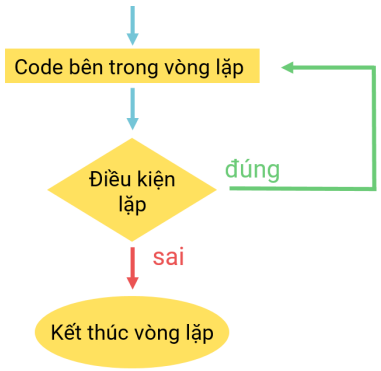
4.3.2 Cơ chế hoạt động

Vòng lặp `do-while` hoạt động theo trình tự:

1. Thực hiện khối lệnh bên trong `do`.
2. Kiểm tra điều kiện `condition`.
3. Nếu điều kiện đúng, quay lại bước 1.
4. Nếu điều kiện sai, kết thúc vòng lặp.

Như vậy, dù `condition` ban đầu là sai, vòng lặp vẫn thực hiện **1 lần**.

4.3.3 Sơ đồ khối vòng lặp `for`



4.3.4 So sánh nhanh với `while`

- `while`: kiểm tra điều kiện **trước** khi chạy ⇒ có thể chạy 0 lần.
- `do-while`: chạy khối lệnh **trước** rồi mới kiểm tra ⇒ luôn chạy ít nhất 1 lần.

4.3.5 Ví dụ cơ bản

Ví dụ 1: In các số từ 1 đến n

```
#include <iostream>
using namespace std;

int main() {
    int n = 4;
    int i = 1;
    do {
        cout << i << "␣";
        i++;
    } while (i <= n);
    return 0;
}
```

Output:

```
1 2 3 4
```

Giải thích

Vòng lặp sẽ:

- In `i`, tăng `i`.
- Sau đó mới kiểm tra `i <= n` để quyết định lặp tiếp hay dừng.

4.3.6 Tình huống đặc trưng của `do-while`

Ví dụ 2: Điều kiện ban đầu sai nhưng vẫn chạy 1 lần

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 10;
    do {
        cout << "Run\n";
        i++;
    } while (i < 5);
    return 0;
}
```

Output:

```
Run
```

Giải thích: Điều kiện `i < 5` là sai ngay từ đầu, nhưng do-while vẫn chạy 1 lần.

4.3.7 Lệnh `break` trong `do-while`

Giống các vòng lặp khác, `break` dùng để thoát vòng lặp ngay lập tức.

Ví dụ 3: Nhập đến khi gặp 28 thì dừng

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    do {
        cout << "Enter␣n:␣";
        cin >> n;
        if (n == 28) break;
        cout << "Try␣again\n";
    } while (true);

    cout << "Finished\n";
    return 0;
}
```

Ví dụ Input:

```
10
5
28
```

Output:

```
Enter n: Try again
Enter n: Try again
Enter n: Finished
```

4.3.8 Lệnh `continue` trong `do-while`

`continue` bỏ qua phần còn lại của lần lặp hiện tại và nhảy đến bước kiểm tra điều kiện.

Ví dụ 4: Chỉ in các số lẻ từ 1 đến n

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 10;
    int i = 0;

    do {
        i++;
        if (i % 2 == 0) continue; // skip even numbers
        cout << i << " ";
    } while (i < n);

    return 0;
}
```

Output:

```
1 3 5 7 9
```

4.3.9 Bài toán điển hình với `do-while`

Bài toán 1: Nhập số nguyên dương (bắt nhập lại nếu sai)

Ý tưởng: Vì người dùng phải nhập ít nhất 1 lần nên `do-while` rất phù hợp.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    do {
        cout << "Enter a positive integer: ";
        cin >> n;
    } while (n <= 0);

    cout << "n = " << n << endl;
    return 0;
}
```

Ví dụ Input:

```
-3
0
5
```

Output:

```
Enter a positive integer: Enter a positive integer: Enter a positive integer: n = 5
```

Bài toán 2: Menu lựa chọn (lặp cho đến khi người dùng chọn thoát)

```cpp
#include <iostream>
using namespace std;

int main() {
    int choice;
    do {
        cout << "1. Print APC\n";
        cout << "2. Print website\n";
        cout << "0. Exit\n";
        cout << "Choose: ";
        cin >> choice;

        if (choice == 1) {
            cout << "APC\n";
        } else if (choice == 2) {
            cout << "sot.umtoj.edu.vn\n";
        } else if (choice == 0) {
            cout << "Bye\n";
        } else {
            cout << "Invalid\n";
        }
    } while (choice != 0);

    return 0;
}
```

Ví dụ Input:

```
2
1
0
```

Output:

```
sot.umtoj.edu.vn
APC
Bye
```

4.3.10 Lưu ý

`do-while` luôn chạy ít nhất một lần, nên không phù hợp nếu bạn cần "có thể chạy 0 lần".

4.4 Bài tập

Bài tập 17. Vòng lặp **link:** https://marisaoj.com/problem/499

Cho hai số nguyên `l, r`. Hãy in ra các số nguyên từ `l` đến r.

Input

Một dòng gồm hai số nguyên `l, r`.

Output

In ra các số nguyên từ `l` đến `r`.

Ví dụ

Dữ liệu vào	Kết quả ra
2 5	2 3 4 5

Cài đặt

```cpp
#include <bits/stdc++.h>
```

```cpp
#define int long long
using namespace std;

signed main(){
    int l,r; cin >> l >> r;
    for(int i = l; i <= r; i++){
        cout << i << "␣";
    }
    return 0;
}
```

Bài tập 18. Số chẵn **link:** https://marisaoj.com/problem/314

Cho số nguyên dương `n`. Hãy in ra các số nguyên dương không vượt quá `n` và chia hết cho `2` theo thứ tự giảm dần.

Input

Một dòng gồm một số nguyên `n`.

Output

In ra các số nguyên dương không vượt quá `n` và chia hết cho `2` theo thứ tự giảm dần.

Điều kiện

$$2 \leq n \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
7	6 4 2

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n; cin >> n;
    for(int i = n; i >= 1; i--){
        if(i % 2 == 0){
            cout << i << "␣";
        }
    }
    return 0;
}
```

Bài tập 19. Giai thừa **link:** https://marisaoj.com/problem/542

Cho số nguyên `n`, hãy tính

$$n! = 1 \times 2 \times \cdots \times n.$$

Input

Một dòng gồm số nguyên `n`.

Output

In ra giá trị `n!`.

Điều kiện

$$1 \leq n \leq 15$$

Ví dụ

Dữ liệu vào	Kết quả ra
4	24

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n; cin >> n;
    int fact = 1;
    for(int i = 1; i <= n; i++){
        fact *= i;
    }
    cout << fact;
    return 0;
}
```

Bài tập 20. Tam giác sao **link:** https://marisaoj.com/problem/16

Cho số nguyên `a`. Hãy in ra một tam giác sao gồm `a` dòng. Dòng thứ nhất gồm `a` ký tự `*`, mỗi dòng tiếp theo giảm đi 1 ký tự `*`. Dòng cuối cùng chỉ có 1 ký tự `*`.

Input

Một dòng gồm số nguyên `a`.

Output

In ra tam giác sao theo mô tả.

Điều kiện

$$1 \leq a \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
5	***** **** *** ** *

Hướng giải.

- Tam giác có `a` dòng.
- Dòng thứ i (từ 1 đến `a`) in ra $a - i + 1$ ký tự `*`.
- Dùng 2 vòng lặp: vòng ngoài chạy theo dòng, vòng trong in dấu sao.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a;
    cin >> a;
    for(int i = a; i >= 1; i--){
        for(int j = 1; j <= i; j++){
            cout << "*";
        }
        cout << "\n";
    }
    return 0;
}
```

Bài tập 21. Gấp giấy **link:** https://marisaoj.com/problem/402

Một tờ giấy khi gấp làm đôi sẽ có độ dày gấp đôi. Bạn có một tờ giấy độ dày `1` cm. Bạn muốn tờ giấy có độ dày ít nhất là `n` cm, hỏi bạn sẽ phải gấp ít nhất bao nhiêu lần?

Input

Một dòng gồm một số nguyên `n`.

Output

In ra một số nguyên là số lần gấp ít nhất.

Điều kiện

$$1 \leq n \leq 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
7	3

Hướng giải.

- Ban đầu độ dày là `1`. Mỗi lần gấp thì độ dày nhân đôi.
- Ta cần tìm số nguyên nhỏ nhất `k` sao cho $2^k \geq n$.
- Mô phỏng: đặt `d = 1`, `ans = 0`. Trong khi `d < n` thì `d *= 2` và `ans++`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    cin >> n;
    int d = 1;
    int ans = 0;
    while(d < n){
        d *= 2;
        ans++;
    }
    cout << ans;
    return 0;
}
```

Bài tập 22. Phân số **link:** https://marisaoj.com/problem/315

Cho hai số nguyên dương `a, b`. Hãy tìm dạng tối giản của phân số $\frac{a}{b}$.

Input

Một dòng gồm hai số nguyên `a, b`.

Output

In ra hai số nguyên, số đầu tiên là tử số, số thứ hai là mẫu số của phân số tối giản.

Điều kiện

$$1 \leq a, b \leq 10^6$$

Ví dụ

Dữ liệu vào	Kết quả ra
6 9	2 3

Hướng giải.

- Phân số $\frac{a}{b}$ tối giản khi $\gcd(a, b) = 1$.
- Tính $g = \gcd(a, b)$.

- Tử số mới: $a' = a/g$, mẫu số mới: $b' = b/g$.
- In ra a' và b'.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b; cin >> a >> b;
    int g = __gcd(a, b);
    cout << a / g << "␣" << b / g;
    return 0;
}
```

Bài tập 23. Ước số **link:** https://marisaoj.com/problem/316

Cho số nguyên dương `n`. Hãy tìm tất cả các ước nguyên dương của `n`.

Input

Một dòng gồm số nguyên `n`.

Output

In ra các ước nguyên dương của `n` theo thứ tự tăng dần.

Điều kiện

$$1 \leq n \leq 10^6$$

Ví dụ

Dữ liệu vào	Kết quả ra
12	1 2 3 4 6 12

Hướng giải.

- Vì cần in theo thứ tự tăng dần, ta duyệt i từ 1 đến n.
- Nếu $n \bmod i = 0$ thì i là một ước của n, in ra ngay.
- Do $n \leq 10^6$, cách này vẫn đủ nhanh trong giới hạn (cách tối ưu sẽ được giới thiệu ở những bài sau).

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++){
        if(n % i == 0){
            cout << i << "␣";
        }
    }
    return 0;
}
```

Bài tập 24. Phép lũy thừa phức tạp **link:** https://marisaoj.com/problem/24

Cho ba số nguyên `a, b, c`. Hãy tính giá trị biểu thức

$$a^b \bmod c.$$

Input

Một dòng gồm ba số nguyên `a, b, c`.

Output

In ra một số nguyên là giá trị của $a^b \bmod c$.

Điều kiện

$$1 \leq a, b, c \leq 10^3$$

Ví dụ

Dữ liệu vào	Kết quả ra
3 5 10	3

Hướng giải.

- Ta không tính trực tiếp a^b (vì có thể rất lớn), mà nhân dần và lấy dư mỗi lần.
- Khởi tạo `res = 1`.
- Lặp `b` lần: `res = (res * a) % c`.
- Vì $b \leq 1000$, cách này chạy rất nhanh. Khi giới hạn lớn hơn ($b \leq 10^{18}$), ta có thể dùng phương pháp **lũy thừa nhanh** (sẽ được giới thiệu ở những bài sau).

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, c;
    cin >> a >> b >> c;

    int res = 1;
    for(int i = 1; i <= b; i++){
        res = (res * a) % c;
    }

    cout << res;
    return 0;
}
```

Bài tập 25. Số nguyên tố **link:** https://marisaoj.com/problem/18

Cho số nguyên `n`. Hãy kiểm tra `n` có phải số nguyên tố không.

Input

Một dòng gồm số nguyên `n`.

Output

In ra `YES` nếu `n` là số nguyên tố, ngược lại in `NO`.

Điều kiện

$$1 \leq n \leq 10^6$$

Ví dụ

Dữ liệu vào	Kết quả ra
7	YES

Hướng giải.

- Số nguyên tố là số lớn hơn 1 và chỉ có đúng 2 ước: 1 và chính nó.
- Nếu $n \leq 1$ thì chắc chắn không phải số nguyên tố.

- Duyệt i từ 2 đến $n-1$:
  - Nếu tồn tại i sao cho $n \bmod i = 0$ thì n không phải số nguyên tố.
  - Nếu không có ước nào thì n là số nguyên tố.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    cin >> n;

    if(n <= 1){
        cout << "NO";
        return 0;
    }

    for(int i = 2; i <= n - 1; i++){
        if(n % i == 0){
            cout << "NO";
            return 0;
        }
    }

    cout << "YES";
    return 0;
}
```

Bài tập 26. Tổng chữ số **link:** https://marisaoj.com/problem/19

Cho số nguyên `n`. Hãy tính tổng các chữ số của `n`.

Input

Một dòng gồm số nguyên `n`.

Output

In ra kết quả của bài toán.

Điều kiện

$$-10^{18} \leq n \leq 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
3218	14

Hướng giải.

- Nếu `n` là số âm, ta lấy giá trị tuyệt đối của `n`.
- Lặp khi `n > 0`:
  - Lấy chữ số cuối bằng `n` mod 10.
  - Cộng vào tổng.
  - Loại bỏ chữ số cuối bằng `n/` $= 10$.
- Trường hợp `n = 0` thì tổng chữ số bằng 0.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;
```

```
4
5   signed main(){
6       int n; cin >> n;
7
8       n = llabs(n); // lay gia tri tuyet doi cua kieu du lieu long long
9       int sum = 0;
10
11      while(n > 0){
12          sum += n % 10;
13          n /= 10;
14      }
15
16      cout << sum;
17      return 0;
18  }
```

Bài tập 27. Fibonacci **link:** https://marisaoj.com/problem/20

In ra số Fibonacci thứ `n`.

Số Fibonacci được định nghĩa:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}.$$

Input

Một dòng gồm số nguyên `n`.

Output

In ra giá trị `F_n`.

Điều kiện

$$1 \leq n \leq 80$$

Ví dụ

Dữ liệu vào	Kết quả ra
4	3

Hướng giải.

- Dựa trực tiếp vào định nghĩa Fibonacci, ta tính tuần tự từ F_0, F_1 lên F_n. Mỗi bước chỉ cần lưu hai giá trị trước đó.

Cài đặt

```
1   #include <bits/stdc++.h>
2   #define int long long
3   using namespace std;
4
5   signed main(){
6       int n;
7       cin >> n;
8
9       if(n == 0){
10          cout << 0;
11          return 0;
12      }
13      if(n == 1){
14          cout << 1;
15          return 0;
16      }
17
18      int f0 = 0, f1 = 1;
19      for(int i = 2; i <= n; i++){
20          int f = f0 + f1;
21          f0 = f1;
22          f1 = f;
```

```
    }

    cout << f1;
    return 0;
}
```

Bài tập 28. Đọc số vĩnh hằng **link:** https://marisaoj.com/problem/22

Nhập vào một dãy số nguyên `n`. Nếu `n` khác 0, hãy in ra giá trị n^5. Nếu `n` bằng 0 thì dừng chương trình.

Input

Một dãy số nguyên, mỗi số trên một dòng.

Output

Với mỗi số `n` khác 0, in ra giá trị n^5 trên một dòng.

Điều kiện

$$-1000 \le n \le 1000, \quad \text{dãy chứa không quá 1000 số}$$

Ví dụ

Dữ liệu vào	Kết quả ra
4 3 12 321 0	1024 243 248832 3408200705601

Hướng giải.

- Đọc lần lượt các số nguyên `n` cho đến khi gặp `n = 0` thì dừng.
- Với mỗi `n` khác 0, tính $n^5 = n \times n \times n \times n \times n$.
- Do $|n| \le 1000$ nên n^5 nằm trong phạm vi của kiểu `long long`.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    while(cin >> n){
        if(n == 0) break;
        int res = n * n * n * n * n;
        cout << res << "\n";
    }
    return 0;
}
```

Bài tập 29. Cực trị **link:** https://marisaoj.com/problem/23

Cho số nguyên `n` và dãy `A` gồm `n` số nguyên. Hãy tìm giá trị lớn nhất và nhỏ nhất của dãy.

Input

Dòng đầu tiên gồm số nguyên `n`.
Dòng thứ hai gồm `n` số nguyên `A_i`.

Output

In ra hai số: giá trị lớn nhất và giá trị nhỏ nhất của dãy (theo đúng thứ tự như ví dụ).

Điều kiện

$$1 \le n \le 1000, \quad |A_i| \le 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
5	999 -2

Hướng giải.

- Đọc `n`, sau đó đọc lần lượt từng phần tử `x` của dãy.
- Khởi tạo `mx` và `mn` bằng phần tử đầu tiên.
- Với mỗi phần tử tiếp theo:
  - Cập nhật `mx = max(mx, x)`.
  - Cập nhật `mn = min(mn, x)`.
- Cuối cùng in ra `mx` và `mn`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    cin >> n;

    int x;
    cin >> x;
    int mx = x, mn = x;

    for(int i = 2; i <= n; i++){
        cin >> x;
        mx = max(mx, x);
        mn = min(mn, x);
    }

    cout << mx << "␣" << mn;
    return 0;
}
```

Bài tập 30. Đảo ngược **link:** https://marisaoj.com/problem/517

Cho hai số nguyên `a, b`. Hãy tìm tổng của chúng và in số này theo thứ tự ngược lại.

Input

Một dòng gồm hai số nguyên `a, b`.

Output

In ra một số nguyên là tổng của `a` và `b` nhưng viết ngược lại. Số in ra không được có chữ số `0` ở đầu.

Ràng buộc

$$1 \leq a, b \leq 10^9$$

Ví dụ

Dữ liệu vào	Kết quả ra
1000 230	321

Hướng giải.

- Tính tổng $s = a + b$.
- Đảo ngược các chữ số của s:
  - Lấy chữ số cuối bằng s mod 10 và ghép vào kết quả.

- Loại bỏ chữ số cuối bằng $s/ = 10$.

- Việc đảo ngược tự nhiên sẽ loại bỏ các chữ số `0` ở đầu.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b;
    cin >> a >> b;
    int s = a + b;

    int rev = 0;
    while(s > 0){
        rev = rev * 10 + (s % 10);
        s /= 10;
    }

    cout << rev;
    return 0;
}
```

Bài tập 31. Đổi nấm **link:** https://marisaoj.com/problem/401

Cửa hàng nấm của Marisa đang có khuyến mãi: đổi `k` cuống nấm sẽ được tặng 1 cây nấm mới. Hiện tại bạn có `n` cây nấm. Mỗi ngày bạn ăn hết 1 cây nấm (tức là tạo ra 1 cuống nấm). Hỏi sau bao nhiêu ngày thì bạn hết nấm?

Input

Một dòng gồm hai số nguyên `n, k`.

Output

In ra một số nguyên là số ngày trước khi bạn hết nấm.

Điều kiện

$$1 \le n \le 10^5, \quad 2 \le k \le 10^5$$

Ví dụ

Dữ liệu vào	Kết quả ra
10 3	14

Hướng giải.

- Mỗi ngày:
  - Nếu còn nấm thì ăn 1 cây: số nấm giảm 1, số cuống tăng 1, số ngày tăng 1.
  - Nếu đủ `k` cuống thì đổi lấy nấm mới: số cuống giảm `k`, số nấm tăng 1.
- Lặp quá trình cho tới khi số nấm bằng 0 thì dừng.
- Vì mỗi ngày chỉ ăn 1 cây, số ngày tăng dần và với ràng buộc đề bài, mô phỏng trực tiếp là đủ.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n, k;
    cin >> n >> k;

    int stems = 0; // so cuong nam
    int days = 0;
```

```
11
12      while(n > 0){
13          // an 1 cay nam
14          n--;
15          stems++;
16          days++;
17
18          // doi cuong lay nam moi (neu du)
19          while(stems >= k){
20              stems -= k;
21              n += 1;
22          }
23      }
24
25      cout << days;
26      return 0;
27  }
```

Bài tập 32. Chữ số 0 tận cùng **link:** https://marisaoj.com/problem/42

Đếm số chữ số `0` tận cùng của `n!`.

Chú ý

$$n! = 1 \times 2 \times \cdots \times n$$

Input

Một số nguyên duy nhất `n`.

Output

In ra một số nguyên là số lượng chữ số `0` tận cùng của `n!`.

Điều kiện

$$1 \leq n \leq 1000$$

Ví dụ

Dữ liệu vào	Kết quả ra
10	2

Hướng giải.

- Một chữ số `0` tận cùng xuất hiện khi trong phép nhân có thừa số $10 = 2 \times 5$.
- Trong $n!$, số lượng thừa số 2 luôn nhiều hơn thừa số 5.
- Vì vậy, số chữ số `0` tận cùng bằng tổng số lần xuất hiện của thừa số 5 trong các số từ 1 đến n.
- Ta lần lượt chia n cho $5, 25, 125, \ldots$ và cộng các thương lại.

Cài đặt

```
1   #include <bits/stdc++.h>
2   #define int long long
3   using namespace std;
4
5   signed main(){
6       int n;
7       cin >> n;
8
9       int cnt = 0;
10      while(n > 0){
11          n /= 5;
12          cnt += n;
13      }
14
15      cout << cnt;
```

```
    return 0;
}
```

Bài tập 33. Thập phân sang nhị phân **link:** https://marisaoj.com/problem/312

Cho một số nguyên dương `n` ở hệ thập phân. Hãy chuyển số này sang hệ nhị phân.

Input

Một dòng gồm một số nguyên `n`.

Output

In ra `n` dưới dạng nhị phân, không được có chữ số `0` đứng đầu.

Điều kiện

$$1 \leq n \leq 10^9$$

Ví dụ

Dữ liệu vào	Kết quả ra
6	110

Hướng giải.

- Hệ nhị phân là hệ cơ số 2.
- Liên tục chia `n` cho 2:
  - Lấy phần dư `n % 2` làm chữ số nhị phân tiếp theo.
  - Cập nhật `n /= 2`.
- Các chữ số thu được theo thứ tự ngược, nên ta ghép ngược lại để được kết quả đúng.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    cin >> n;

    string bin = "";
    while(n > 0){
        bin = char('0' + (n % 2)) + bin;
        n /= 2;
    }

    cout << bin;
    return 0;
}
```

Bài tập 34. Nhị phân sang thập phân **link:** https://marisaoj.com/problem/313

Cho một số `n` dưới dạng nhị phân. Hãy chuyển số này sang hệ thập phân.

Input

Một dòng gồm một số nguyên `n` dưới dạng nhị phân.

Output

In ra `n` dưới dạng thập phân.

Điều kiện

Đảm bảo giá trị thập phân của `n` là số nguyên dương và không vượt quá $2^{31} - 1$.

Ví dụ

Dữ liệu vào	Kết quả ra
110	6

Hướng giải.

- Mỗi chữ số nhị phân có giá trị là 0 hoặc 1.
- Duyệt chuỗi từ trái sang phải:
  - Nhân kết quả hiện tại với 2.
  - Cộng thêm chữ số hiện tại (`'0'` hoặc `'1'`) sau khi đổi sang số.
- Cách này mô phỏng đúng quy tắc chuyển từ hệ nhị phân sang hệ thập phân.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    string n;
    cin >> n;

    int res = 0;
    for(int i = 0; i < (int)n.size(); i++){
        int bit = n[i] - '0';
        res = res * 2 + bit;
    }

    cout << res;
    return 0;
}
```

Bài tập 35. Phép chia **link:** https://marisaoj.com/problem/417

Cho ba số nguyên `a, b, k`. Hãy in ra chữ số thứ `k` sau dấu thập phân của phép chia $\frac{a}{b}$.

Input

Một dòng gồm ba số nguyên `a, b, k`.

Output

In ra một chữ số (từ 0 đến 9) là chữ số thứ `k` sau dấu thập phân.

Điều kiện

$$1 \le a, b, k \le 10^5$$

Ví dụ

Dữ liệu vào	Kết quả ra
22 7 5	5

Hướng giải.

- Ta không cần tính số thực, chỉ cần mô phỏng phép chia lấy phần thập phân.
- Gọi $r = a \bmod b$ là số dư sau phần nguyên.
- Với mỗi vị trí sau dấu phẩy:
  - Nhân số dư lên 10: $r = r \times 10$.
  - Chữ số tiếp theo là $\left\lfloor \frac{r}{b} \right\rfloor$.
  - Cập nhật số dư: $r = r \bmod b$.

- Lặp đúng `k` lần, chữ số ở lần thứ `k` chính là đáp án.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int a, b, k;
    cin >> a >> b >> k;

    int r = a % b;
    int digit = 0;

    for(int i = 1; i <= k; i++){
        r *= 10;
        digit = r / b;
        r %= b;
    }

    cout << digit;
    return 0;
}
```

Bài tập 36. Bộ nghiệm **link:** https://marisaoj.com/problem/425

Cho số nguyên dương `n`. Hãy đếm số lượng bộ bốn số nguyên dương

$$x_1 < x_2 < x_3 < x_4$$

sao cho:

$$x_1 + x_2 + x_3 + x_4 = n.$$

Input

Một dòng gồm một số nguyên dương `n`.

Output

In ra một số nguyên là số lượng bộ số thỏa mãn điều kiện.

Điều kiện

$$1 \leq n \leq 50$$

Ví dụ

Dữ liệu vào	Kết quả ra
10	1

Hướng giải.

- Duyệt bốn số x_1, x_2, x_3, x_4 sao cho:
$$1 \leq x_1 < x_2 < x_3 < x_4$$
- Với mỗi bộ, kiểm tra nếu tổng bằng `n` thì tăng biến đếm.
- Điều kiện $x_1 < x_2 < x_3 < x_4$ được đảm bảo bằng cách:
  - x_1 chạy từ 1
  - x_2 chạy từ $x_1 + 1$
  - x_3 chạy từ $x_2 + 1$
  - x_4 chạy từ $x_3 + 1$

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

signed main(){
    int n;
    cin >> n;

    int cnt = 0;
    for(int x1 = 1; x1 <= n; x1++){
        for(int x2 = x1 + 1; x2 <= n; x2++){
            for(int x3 = x2 + 1; x3 <= n; x3++){
                for(int x4 = x3 + 1; x4 <= n; x4++){
                    if(x1 + x2 + x3 + x4 == n){
                        cnt++;
                    }
                }
            }
        }
    }

    cout << cnt;
    return 0;
}
```

CHƯƠNG 5

HÀM VÀ ĐỆ QUY

Contents

5.1 Hàm (Function) trong C++

5.1.1 Vì sao phải học hàm? (động cơ + vấn đề cần giải quyết)

Ở các bài đầu, ta thường viết toàn bộ chương trình trong `main`. Cách này chạy được với bài nhỏ, nhưng khi bài toán lớn dần sẽ gặp 4 vấn đề:

- **Code dài, khó đọc:** bạn nhìn vào `main` không biết đoạn nào làm nhiệm vụ gì.
- **Khó sửa lỗi:** lỗi xuất hiện, bạn phải dò cả một "đống" lệnh.
- **Lặp lại code:** cùng một thao tác phải viết lại nhiều lần.
- **Khó mở rộng:** thêm chức năng mới dễ làm hỏng chức năng cũ.

Hàm ra đời để giải quyết các vấn đề trên bằng cách **chia nhỏ chương trình thành các khối có tên**, mỗi khối làm một việc.

5.1.2 Khái niệm hàm

Trong lập trình, **hàm (function)** là một **đơn vị chương trình độc lập**, được thiết kế để thực hiện một nhiệm vụ xác định. Mỗi hàm có thể nhận dữ liệu đầu vào, xử lý dữ liệu đó, và (tuỳ trường hợp) trả về một kết quả cho nơi gọi hàm.

Về mặt khái niệm, một hàm bao gồm ba giai đoạn:

- **Tiếp nhận dữ liệu** thông qua các **tham số (parameters)**;
- **Thực hiện xử lý** thông qua một dãy câu lệnh;
- **Trả kết quả** thông qua **giá trị trả về (return value)** hoặc kết thúc thực thi.

Do đó, trong ý nghĩa trừu tượng, một hàm có thể được xem là một ánh xạ:

$$f : \text{Input} \to \text{Output}$$

Trong C++, nhiều hàm chuẩn đã được cung cấp sẵn, ví dụ:

- `abs(x)`: ánh xạ một số nguyên sang trị tuyệt đối của nó;
- `sqrt(x)`: ánh xạ một số thực không âm sang căn bậc hai tương ứng.

Việc tự xây dựng hàm cho phép người lập trình định nghĩa các phép xử lý phù hợp với bài toán cụ thể, đồng thời nâng cao tính tổ chức và tái sử dụng của chương trình.

5.1.3 Các thành phần cấu thành của một hàm

Mỗi hàm trong C++ được đặc trưng bởi ba thành phần cơ bản:

1. **Tên hàm**: định danh hàm trong chương trình và phản ánh chức năng của hàm;
2. **Danh sách tham số**: mô tả các dữ liệu đầu vào mà hàm tiếp nhận;
3. **Kiểu giá trị trả về**: xác định kiểu dữ liệu của kết quả mà hàm trả về sau khi thực thi.

Trong trường hợp hàm không cần trả về kết quả, kiểu trả về được khai báo là `void`.

5.1.4 Cú pháp định nghĩa hàm

Cú pháp tổng quát để định nghĩa một hàm trong C++ như sau:

```
return_type function_name(parameter_list) {
    // function body
    return value; // if return_type is not void
}
```

Trong đó:

- **return_type**: kiểu dữ liệu của giá trị mà hàm trả về. Các kiểu thường gặp gồm `int`, `long long`, `double`, `bool`, `string`. Nếu hàm không trả về giá trị, kiểu trả về là `void`.
- **function_name**: tên hàm, tuân theo quy tắc đặt tên định danh của C++, và nên phản ánh rõ chức năng của hàm.
- **parameter_list**: danh sách các tham số, mỗi tham số gồm **kiểu dữ liệu** và **tên biến**.

5.1.5 Ví dụ 1: Hàm không có tham số và không trả về

Xét hàm chỉ thực hiện một thao tác mà không cần dữ liệu đầu vào và không trả về kết quả:

```
void printAPC() {
    cout << "APC␣-␣sot.umtoj.edu.vn\n";
}

int main() {
    printAPC();
    printAPC();
}
```

Giải thích:

- Hàm `printAPC` có kiểu trả về `void`, do đó không trả về giá trị.
- Mỗi lần gọi hàm, nội dung bên trong hàm được thực thi đầy đủ.

5.1.6 Ví dụ 2: Hàm có tham số và có giá trị trả về

Xét hàm thực hiện phép cộng hai số nguyên:

```
int add(int a, int b) {
    return a + b;
}

int main() {
    int x = add(3, 5);
    cout << x;
}
```

Giải thích:

- `a`, `b` là các tham số hình thức của hàm.
- Khi gọi `add(3,5)`, các giá trị 3 và 5 được truyền vào tham số tương ứng.
- Hàm trả về giá trị 8, được gán cho biến `x`.

5.1.7 Tham số và đối số

- **Tham số (parameter)** là các biến được khai báo trong định nghĩa hàm.
- **Đối số (argument)** là các giá trị cụ thể được truyền vào khi gọi hàm.

```
int add(int a, int b) { // parameters
    return a + b;
}

int main() {
    cout << add(10, 20); // arguments
}
```

5.1.8 Luồng thực thi khi gọi hàm

Khi chương trình gọi một hàm, luồng điều khiển được chuyển tạm thời từ hàm gọi sang hàm được gọi.

Xét đoạn chương trình sau:

```
int add(int a, int b) {
    return a + b;
}

int main() {
    int x = add(2, 3);
    cout << x;
}
```

Trình tự thực thi diễn ra như sau:

1. Chương trình đang thực thi trong hàm `main`.
2. Khi gặp lời gọi `add(2,3)`, luồng điều khiển chuyển sang hàm `add`.
3. Các tham số `a` và `b` lần lượt nhận giá trị 2 và 3.
4. Hàm `add` thực hiện phép cộng và trả về kết quả.
5. Luồng điều khiển quay lại `main`, tiếp tục thực thi câu lệnh tiếp theo.

5.1.9 Lệnh `return`

Lệnh `return` có hai vai trò quan trọng:

- Trả về một giá trị cho nơi gọi hàm (nếu hàm không phải `void`);
- Kết thúc ngay lập tức quá trình thực thi của hàm.

```
int absVal(int x) {
    if (x < 0) return -x;
    return x;
}
```

5.1.10 Phạm vi biến và biến cục bộ

Các biến được khai báo bên trong hàm được gọi là **biến cục bộ**, chỉ tồn tại trong suốt thời gian thực thi của hàm đó.

```
void test() {
    int x = 5;
    cout << x << "\n";
}

int main() {
    test();
}
```

Ngược lại, các biến được khai báo bên ngoài mọi hàm là **biến toàn cục** và có phạm vi sử dụng rộng hơn.

5.2 Cơ chế truyền tham số và tham chiếu trong C++

5.2.1 Mở đầu

Ở các phần trước, ta đã biết cách định nghĩa và gọi hàm trong C++. Tuy nhiên, một vấn đề cốt lõi ảnh hưởng trực tiếp đến hành vi của chương trình là: **cách đối số được truyền vào hàm thông qua tham số**.

Cơ chế truyền tham số quyết định liệu những thay đổi bên trong hàm có được giữ lại sau khi hàm kết thúc hay không. Để hiểu chính xác vấn đề này, ta cần làm rõ vai trò của **toán tử `&`** và khái niệm **tham chiếu (reference)**.

5.2.2 Toán tử `&` trong C++

Trong C++, ký hiệu `&` có **hai ý nghĩa hoàn toàn khác nhau** tùy theo ngữ cảnh sử dụng.

5.2.2.1 Toán tử lấy địa chỉ

Khi `&` được đặt trước một biến trong biểu thức, nó là toán tử **lấy địa chỉ**, dùng để truy xuất địa chỉ bộ nhớ của biến đó.

```
int x = 5;
cout << &x;
```

Output (ví dụ):

```
0x6ffe1c
```

Giá trị in ra là địa chỉ ô nhớ mà biến `x` đang quản lý, thường được biểu diễn dưới dạng số hệ thập lục phân.

5.2.2.2 Toán tử khai báo tham chiếu

Khi `&` xuất hiện trong khai báo biến, giữa kiểu dữ liệu và tên biến, nó dùng để khai báo một **tham chiếu**.

```
int a = 10;
int &r = a;
r = 20;
cout << a;
```

Output:

```
20
```

Trong trường hợp này, `r` không tạo ra một biến mới, mà trở thành một định danh khác cùng quản lý ô nhớ với `a`.

5.2.3 Khái niệm tham chiếu

Tham chiếu (reference) là một cơ chế ngôn ngữ cho phép liên kết một định danh với một đối tượng đã tồn tại. Một số tính chất quan trọng của tham chiếu:

- Tham chiếu phải được khởi tạo ngay khi khai báo;
- Sau khi đã liên kết, tham chiếu không thể đổi sang đối tượng khác;
- Mọi thao tác trên tham chiếu đều tác động trực tiếp lên đối tượng được tham chiếu.

5.2.4 Truyền tham trị (Pass by value)

Khi tham số của hàm không có ký hiệu `&`, C++ sử dụng cơ chế **truyền tham trị**. Theo đó, giá trị của đối số được sao chép vào tham số của hàm.

```
#include <iostream>
using namespace std;

void thaydoi(int n){
    cout << "Dia␣chi␣cua␣n:␣" << &n << endl;
    n += 100;
    cout << "Gia␣tri␣cua␣n␣trong␣ham:␣" << n << endl;
}

int main(){
    int m = 1000;
    cout << "Dia␣chi␣cua␣m:␣" << &m << endl;
    thaydoi(m);
    cout << "Gia␣tri␣cua␣m␣sau␣khi␣ham␣ket␣thuc:␣" << m << endl;
    return 0;
}
```

Output (ví dụ):

```
Dia chi cua m: 0x6ffe1c
Dia chi cua n: 0x6ffdf0
Gia tri cua n trong ham: 1100
Gia tri cua m sau khi ham ket thuc: 1000
```

Phân tích:

- `m` và `n` có hai địa chỉ bộ nhớ khác nhau;
- `n` chỉ là bản sao giá trị của `m`;
- Thay đổi `n` không ảnh hưởng đến `m`.

Kết luận: Truyền tham trị tạo ra các bản sao độc lập, do đó không thể dùng để thay đổi trực tiếp đối số sau khi hàm kết thúc.

5.2.5 Truyền tham chiếu (Pass by reference)

Khi tham số được khai báo dưới dạng tham chiếu, tham số và đối số cùng quản lý một ô nhớ trong bộ nhớ.

```
#include <iostream>
using namespace std;

void thaydoi(int &n){
    cout << "Dia␣chi␣cua␣n:␣" << &n << endl;
    n += 100;
}

```

```
int main(){
    int m = 1000;
    cout << "Dia chi cua m: " << &m << endl;
    thaydoi(m);
    cout << "Gia tri cua m sau khi ham ket thuc: " << m << endl;
    return 0;
}
```

Output (ví dụ):

```
Dia chi cua m: 0x6ffe1c
Dia chi cua n: 0x6ffe1c
Gia tri cua m sau khi ham ket thuc: 1100
```

Phân tích:

- `n` là tham chiếu trực tiếp tới `m`;
- Mọi thay đổi trên `n` đều tác động lên `m`;
- Sau khi hàm kết thúc, giá trị thay đổi được giữ lại.

5.2.6 Tham chiếu hằng (Const reference)

Trong trường hợp hàm chỉ cần đọc dữ liệu, việc sử dụng `const` reference giúp tăng hiệu năng mà vẫn đảm bảo an toàn.

```
#include <iostream>
#include <string>
using namespace std;

int length(const string &s){
    return s.size();
}

int main(){
    cout << length("APC");
    return 0;
}
```

Output:

```
3
```

Nhận xét:

- Không xảy ra sao chép chuỗi;
- Hàm không được phép thay đổi dữ liệu của đối số.

5.2.7 Ví dụ tổng hợp: Hoán đổi hai biến

```
#include <iostream>
using namespace std;

void swap1(int a, int b){
    int tmp = a;
    a = b;
    b = tmp;
}

void swap2(int &a, int &b){
    int tmp = a;
    a = b;
```

```
    b = tmp;
}

int main(){
    int x = 100, y = 200;
    swap1(x, y);
    cout << x << "␣" << y << endl;
    swap2(x, y);
    cout << x << "␣" << y << endl;
    return 0;
}
```

Output:

```
100 200
200 100
```

5.2.8 Tổng kết

- Toán tử `&` có hai vai trò: lấy địa chỉ và khai báo tham chiếu;
- Truyền tham trị tạo bản sao, không ảnh hưởng đối số;
- Truyền tham chiếu cho phép hàm thay đổi trực tiếp biến được truyền vào;
- `const reference` là lựa chọn tối ưu khi chỉ cần đọc dữ liệu.

5.3 Đối số mặc định của hàm trong C++

5.3.1 Khái niệm

Đối số mặc định (default argument) là giá trị được gán sẵn cho một tham số ngay tại thời điểm **khai báo hoặc định nghĩa hàm**. Giá trị này sẽ được sử dụng tự động khi lời gọi hàm **không cung cấp đối số tương ứng**.

Nói cách khác, đối số mặc định cho phép một hàm **được gọi với số lượng đối số ít hơn số tham số đã khai báo**, miễn là các tham số bị thiếu đã có giá trị mặc định.

5.3.2 Cơ chế hoạt động

Đối số mặc định được xử lý tại **thời điểm biên dịch (compile-time)**, không phải tại thời điểm chạy chương trình.

Khi biên dịch, trình biên dịch sẽ:

- kiểm tra số lượng đối số trong lời gọi hàm;
- nếu thiếu, tự động chèn các giá trị mặc định tương ứng;
- sau đó sinh mã như một lời gọi hàm đầy đủ tham số.

Vì vậy, **đối số mặc định không phải là một cơ chế động** và không liên quan đến việc kiểm tra khi chương trình đang chạy.

5.3.3 Ví dụ cơ bản về đối số mặc định

```
#include <iostream>
using namespace std;

int find(int a, int b, int c = 3, int d = 4){
```

```
    cout << a << "␣" << b << "␣" << c << "␣" << d << endl;
    return a + b + c + d;
}

int main(){
    cout << find(1, 2) << endl;
    cout << find(1, 2, 10) << endl;
    cout << find(1, 2, 10, 20) << endl;
    return 0;
}
```

Output:

```
1 2 3 4
10
1 2 10 4
17
1 2 10 20
33
```

5.3.4 Phân tích chi tiết

- Lời gọi `find(1, 2)`:
  - thiếu đối số thứ 3 và 4;
  - `c` nhận giá trị mặc định là 3;
  - `d` nhận giá trị mặc định là 4.
- Lời gọi `find(1, 2, 10)`:
  - đối số thứ 3 được cung cấp nên ghi đè giá trị mặc định;
  - đối số thứ 4 vẫn dùng giá trị mặc định là 4.
- Lời gọi `find(1, 2, 10, 20)`:
  - tất cả đối số đều được truyền đầy đủ;
  - không sử dụng bất kỳ giá trị mặc định nào.

5.3.5 Quy tắc bắt buộc của đối số mặc định

Trong C++, các tham số có đối số mặc định **phải nằm ở cuối danh sách tham số**. Nói cách khác:

Sau khi một tham số có giá trị mặc định xuất hiện, mọi tham số phía sau nó đều phải có giá trị mặc định.

5.3.6 Ví dụ sai và lỗi biên dịch

```
int find(int a, int b, int c = 3, int d){
    return a + b + c + d;
}
```

Lỗi biên dịch:

```
error: default argument missing for parameter 4 of 'int find(int, int, int, int)'
```

Giải thích:

- `c` có giá trị mặc định;
- `d` đứng sau `c` nhưng không có giá trị mặc định;
- trình biên dịch không thể suy ra giá trị cho `d` khi lời gọi hàm thiếu đối số.

5.3.7 Đối số mặc định và ghi đè giá trị

Đối số mặc định **không phải là hằng số cố định**. Nếu lời gọi hàm cung cấp đối số, giá trị mặc định sẽ **bị ghi đè hoàn toàn**.

```
int f(int x = 10){
    return x * 2;
}

int main(){
    cout << f() << endl;
    cout << f(7) << endl;
}
```

Output:

```
20
14
```

5.3.8 Một số lưu ý quan trọng

- Đối số mặc định khác với việc truyền hằng số;
- Giá trị mặc định được gán tại thời điểm biên dịch;
- Đối số mặc định thường được khai báo trong prototype (đặc biệt khi dùng nhiều file);
- Không nên lạm dụng đối số mặc định vì có thể làm hàm khó đọc và khó bảo trì.

5.3.9 Tổng kết

- Đối số mặc định cho phép gọi hàm với số đối số ít hơn số tham số;
- Giá trị mặc định được sử dụng khi đối số tương ứng bị thiếu;
- Các tham số có đối số mặc định phải nằm ở cuối danh sách;
- Nếu lời gọi hàm cung cấp đối số, giá trị mặc định sẽ bị ghi đè.

5.4 Hàm đệ quy (Recursion) trong C++

5.4.1 Khái niệm và động cơ sử dụng

Đệ quy (recursion) là kỹ thuật trong đó một hàm **tự gọi lại chính nó** (trực tiếp hoặc gián tiếp) để giải quyết bài toán. Tư tưởng cốt lõi của đệ quy là: **chia bài toán lớn thành các bài toán con cùng dạng** nhưng có kích thước nhỏ hơn, cho đến khi bài toán con trở nên đủ nhỏ để giải trực tiếp.

Một lời giải đệ quy đúng luôn phải có hai thành phần:

- **Trường hợp cơ sở (base case):** điều kiện dừng, xử lý trực tiếp, không gọi tiếp.
- **Bước đệ quy (recursive step):** chuyển bài toán hiện tại thành bài toán nhỏ hơn và gọi lại hàm.

5.4.2 Mô hình tư duy đệ quy: công thức truy hồi

Khi viết đệ quy, ta thường biểu diễn bài toán bằng một **công thức truy hồi**:

$$F(n) = \begin{cases} \text{giá trị trực tiếp} & \text{nếu } n \text{ thuộc base case} \\ \text{kết hợp từ } F(\text{bài toán nhỏ hơn}) & \text{ngược lại} \end{cases}$$

Điểm quan trọng: trong bước đệ quy, tham số phải "tiến gần" về base case, nếu không chương trình sẽ rơi vào **đệ quy vô hạn**.

5.4.3 Cơ chế thực thi: ngăn xếp lời gọi (Call Stack)

Mỗi lần một hàm được gọi, hệ thống tạo một **khung ngăn xếp (stack frame)** mới để lưu:

- tham số của lần gọi,
- biến cục bộ,
- địa chỉ quay về sau khi hàm kết thúc (return address).

Trong đệ quy, các stack frame được xếp chồng lên nhau theo thứ tự gọi. Khi gặp base case và bắt đầu `return`, chương trình "tháo" dần các frame (theo cơ chế LIFO: vào sau ra trước) để trả kết quả về cho các lần gọi trước đó.

5.4.4 Cú pháp tổng quát

```
return_type f(parameters) {
    if (base_case_condition) {
        // base case
        return base_value;
    }
    // recursive step
    return combine( f(smaller_problem) );
}
```

5.4.5 Ví dụ 1: Tính giai thừa

Định nghĩa:

$$n! = \begin{cases} 1 & \text{nếu } n = 0 \text{ hoặc } n = 1 \\ n \cdot (n-1)! & \text{nếu } n \geq 2 \end{cases}$$

```
#include <iostream>
using namespace std;

long long fact(int n) {
    if (n <= 1) return 1;           // base case
    return 1LL * n * fact(n - 1);  // recursive step
}

int main() {
    cout << fact(5) << "\n";
    return 0;
}
```

Output:

```
120
```

5.4.6 Ví dụ 2: Dãy Fibonacci

Định nghĩa:

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2) \ (n \geq 2)$$

```
#include <iostream>
using namespace std;

long long fib(int n) {
```

```
    if (n == 0) return 0;                // base case
    if (n == 1) return 1;                // base case
    return fib(n - 1) + fib(n - 2);      // recursive step
}

int main() {
    cout << fib(10) << "\n";
    return 0;
}
```

Output:

```
55
```

Lưu ý quan trọng: Cách viết Fibonacci đệ quy thuần tuý có độ phức tạp thời gian rất lớn (do lặp lại nhiều phép tính). Với n lớn, cần dùng quy hoạch động (DP) hoặc memoization.

5.4.7 Ví dụ 3: Tính tổng 1 đến n

Công thức:

$$S(n) = 1 + 2 + \cdots + n = \begin{cases} 0 & \text{nếu } n = 0 \\ n + S(n-1) & \text{nếu } n \geq 1 \end{cases}$$

```
#include <iostream>
using namespace std;

long long sum1ToN(int n) {
    if (n == 0) return 0;
    return 1LL * n + sum1ToN(n - 1);
}

int main() {
    cout << sum1ToN(10) << "\n";
    return 0;
}
```

Output:

```
55
```

5.4.8 Đệ quy có tham số tích luỹ (Accumulator) và tối ưu đuôi

Một số bài toán cho phép viết theo dạng **đệ quy đuôi (tail recursion)**: lời gọi đệ quy là thao tác cuối cùng của hàm. Điều này giúp logic rõ ràng và đôi khi trình biên dịch có thể tối ưu.

```
#include <iostream>
using namespace std;

long long sumTail(int n, long long acc) {
    if (n == 0) return acc;
    return sumTail(n - 1, acc + n);
}

int main() {
    cout << sumTail(10, 0) << "\n";
    return 0;
}
```

Output:

```
55
```

5.4.9 Bài toán điển hình: Ước chung lớn nhất (Euclid)

Thuật toán Euclid dựa trên tính chất:

$$\gcd(a, b) = \gcd(b, a \bmod b), \quad \gcd(a, 0) = a$$

```cpp
#include <iostream>
using namespace std;

long long gcd(long long a, long long b) {
    if (b == 0) return a;         // base case
    return gcd(b, a % b);         // recursive step
}

int main() {
    cout << gcd(48, 18) << "\n";
    return 0;
}
```

Output:

```
6
```

5.4.10 Lỗi thường gặp khi viết đệ quy

- **Quên base case** hoặc base case sai ⇒ đệ quy vô hạn.
- **Bài toán không giảm kích thước** ⇒ không tiến về điều kiện dừng.
- **Tràn ngăn xếp (stack overflow)** khi độ sâu đệ quy quá lớn (ví dụ gọi 10^6 lần).
- **Lặp tính toán** như Fibonacci thuần tuý ⇒ thời gian chạy rất lớn.

5.4.11 Khi nào nên dùng đệ quy?

Đệ quy phù hợp khi bài toán có cấu trúc phân rã tự nhiên, ví dụ:

- duyệt cây, duyệt đồ thị theo DFS,
- sinh hoán vị, tổ hợp, quay lui (backtracking),
- chia để trị (merge sort, quick sort),
- các công thức truy hồi rõ ràng (gcd, lũy thừa nhanh).

5.4.12 Tổng kết

- Hàm đệ quy là hàm tự gọi lại chính nó để giải bài toán bằng phân rã.
- Đệ quy đúng phải có: base case và bước đệ quy làm nhỏ bài toán.
- Cơ chế thực thi dựa trên call stack, nên có nguy cơ tràn ngăn xếp nếu gọi quá sâu.
- Cần cảnh giác với các đệ quy gây lặp tính toán; khi cần, dùng memoization/DP.

5.5 Bài tập

Bài tập 37. Số Fibonacci **link:**

Cho số nguyên không âm `n`. Hãy tính số Fibonacci thứ `n` với quy ước:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \ (n \geq 2).$$

Input

Một dòng chứa một số nguyên `n`.

Output

In ra giá trị `F_n`.

Điều kiện

$$0 \leq n \leq 40$$

Ví dụ

Dữ liệu vào	Kết quả ra
12	144

Hướng giải.

- Xác định bài toán cơ sở:
$$F(0) = 0, \quad F(1) = 1.$$
- Với $n \geq 2$ áp dụng công thức truy hồi:
$$F(n) = F(n-1) + F(n-2).$$
- Viết hàm đệ quy `F(n)` với hai trường hợp **base case** và **recursive step**.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

int F(int n){
    if(n == 0 || n == 1) return n;
    return F(n - 1) + F(n - 2);
}

signed main(){
    int n;
    cin >> n;
    cout << F(n);
    return 0;
}
```

Bài tập 38. Tổ hợp chập k của n **link:**

Cho hai số nguyên `n, k`. Hãy tính tổ hợp chập `k` của `n`:

$$C(n,k) = \binom{n}{k}.$$

Ta có:

$$C(n,0) = 1, \quad C(n,n) = 1, \quad C(n,k) = C(n-1,k-1) + C(n-1,k) \ (0 < k < n).$$

Input

Một dòng gồm hai số nguyên `n, k`.

Output

In ra giá trị `C(n,k)`.

Điều kiện

$$0 \leq k \leq n \leq 20$$

Ví dụ

Dữ liệu vào	Kết quả ra
12 2	66

Hướng giải.

- **Base case:** nếu `k = 0` hoặc `k = n` thì trả về `1`.
- **Bước đệ quy:**
$$C(n,k) = C(n-1,k-1) + C(n-1,k).$$
- Viết hàm đệ quy `C(n,k)` theo đúng công thức trên.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

int C(int n, int k){
    if(k == 0 || k == n) return 1;
    return C(n - 1, k - 1) + C(n - 1, k);
}

signed main(){
    int n, k;
    cin >> n >> k;
    cout << C(n, k);
    return 0;
}
```

Bài tập 39. Chuyển số thập phân sang nhị phân **link:**

Cho số nguyên không âm `n`. Hãy in biểu diễn **nhị phân** của `n` (không in số 0 thừa ở đầu).

Input

Một dòng gồm một số nguyên `n`.

Output

In ra biểu diễn nhị phân của `n`.

Điều kiện

$$0 \leq n \leq 10^{12}$$

Ví dụ

Dữ liệu vào	Kết quả ra
37	100101

Hướng giải.

- Sử dụng cách chuyển cơ số: chia `n` cho 2, lưu lại số dư.
- **Base case:** nếu `n < 2` thì in trực tiếp `n` (0 hoặc 1).
- **Bước đệ quy:**
  - Gọi đệ quy với `n / 2`.

- Sau khi quay về, in `n % 2`.

- Thứ tự gọi trước, in sau đảm bảo các bit được in theo đúng thứ tự từ trái sang phải.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

void dec_to_bin(long long n){
    if(n < 2){
        cout << n;
    }else{
        dec_to_bin(n / 2);
        cout << (n % 2);
    }
}

signed main(){
    long long n;
    cin >> n;
    if(n == 0){
        cout << 0;
    }else{
        dec_to_bin(n);
    }
    return 0;
}
```

Bài tập 40. Chuyển số thập phân sang hệ 16 **link:**

Cho số nguyên không âm `n`. Hãy in biểu diễn **hệ 16** (hexadecimal) của `n`, sử dụng các ký tự `0..9, A..F`.

Input

Một dòng gồm một số nguyên `n`.

Output

In ra biểu diễn hệ 16 của `n`.

Điều kiện

$$0 \leq n \leq 10^{12}$$

Ví dụ

Dữ liệu vào	Kết quả ra
762	2FA

Hướng giải.

- Dùng thuật toán chia cho 16, lưu số dư.

- **Base case:** nếu `n < 16`, in một ký tự duy nhất:

  - `0..9` giữ nguyên,

  - `10..15` chuyển thành `A..F` bằng `char(n + 55)`.

- **Bước đệ quy:** gọi hàm với `n / 16`, sau đó in phần dư `n % 16`.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

void dec_to_hex(long long n){
    if(n < 16){
```

```
        if(n < 10) cout << n;
        else cout << char(n + 55); // 10 -> 'A', 11 -> 'B', ...
    }else{
        dec_to_hex(n / 16);
        int r = n % 16;
        if(r < 10) cout << r;
        else cout << char(r + 55);
    }
}

signed main(){
    long long n;
    cin >> n;
    if(n == 0){
        cout << 0;
    }else{
        dec_to_hex(n);
    }
    return 0;
}
```

Bài tập 41. Đếm số chữ số **link:**

Cho số nguyên dương `n`. Hãy đếm số chữ số của `n`.

Input

Một dòng gồm một số nguyên dương `n`.

Output

In ra số chữ số của `n`.

Điều kiện

$$1 \le n \le 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
28282828	8

Hướng giải.

- **Base case:** nếu `n < 10` thì chỉ có 1 chữ số.
- **Bước đệ quy:** với `n` $\ge$ `10`:

$$D(n) = 1 + D\left(\left\lfloor \frac{n}{10} \right\rfloor\right).$$

- Mỗi lần gọi đệ quy bỏ đi chữ số hàng đơn vị, đồng thời cộng thêm 1.

Cài đặt

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

int D(long long n){
    if(n < 10) return 1;
    return 1 + D(n / 10);
}

signed main(){
    long long n;
    cin >> n;
    cout << D(n);
    return 0;
}
```

Bài tập 42. Tổng chữ số **link:**

Cho số nguyên dương `n`. Hãy tính tổng các chữ số của `n`.

Input

Một dòng gồm một số nguyên dương `n`.

Output

In ra tổng các chữ số của `n`.

Điều kiện

$$1 \leq n \leq 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
28282828	40

Hướng giải.

- **Base case:** nếu `n < 10` thì tổng chữ số là `n`.
- **Bước đệ quy:**

$$S(n) = (n \bmod 10) + S\left(\left\lfloor \frac{n}{10} \right\rfloor\right).$$

- Mỗi bước tách chữ số cuối cùng và cộng với kết quả của phần còn lại.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

int S(long long n){
    if(n < 10) return n;
    return (n % 10) + S(n / 10);
}

signed main(){
    long long n;
    cin >> n;
    cout << S(n);
    return 0;
}
```

Bài tập 43. Chữ số lớn nhất **link:**

Cho số nguyên dương `n`. Hãy tìm chữ số lớn nhất trong `n`.

Input

Một dòng gồm một số nguyên dương `n`.

Output

In ra chữ số lớn nhất trong `n`.

Điều kiện

$$1 \leq n \leq 10^{18}$$

Ví dụ

Dữ liệu vào	Kết quả ra
12349567	9

Hướng giải.

- **Base case:** nếu `n < 10` thì kết quả là `n`.
- **Bước đệ quy:**

$$F(n) = \max\left(n \bmod 10,\ F\left(\left\lfloor \frac{n}{10} \right\rfloor\right)\right).$$

- So sánh chữ số cuối cùng với kết quả lớn nhất của phần còn lại.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;

int F(long long n){
    if(n < 10) return n;
    int tmp = F(n / 10);
    int last = n % 10;
    return (last > tmp ? last : tmp);
}

signed main(){
    long long n;
    cin >> n;
    cout << F(n);
    return 0;
}
```

CHƯƠNG 6

MẢNG MỘT CHIỀU VÀ CẤU TRÚC DỮ LIỆU `vector`

Contents

6.1 Mảng một chiều (`array`)

6.1.1 Định nghĩa và bản chất lưu trữ trong bộ nhớ

Trong lập trình, dữ liệu thường xuất hiện dưới dạng **danh sách** (list) các giá trị cùng kiểu: dãy điểm số, dãy nhiệt độ theo ngày, dãy toạ độ, ... **Mảng một chiều** là cấu trúc dữ liệu cơ bản để lưu trữ các danh sách như vậy.

- **Định nghĩa:** Mảng một chiều là cấu trúc dữ liệu dùng để lưu trữ nhiều phần tử **cùng kiểu dữ liệu**, các phần tử được đặt **liên tiếp nhau** trong bộ nhớ và được truy cập thông qua **chỉ số (index)**.
- **Bản chất:** Khi khai báo `int a[5];` thì chương trình cấp phát **một khối ô nhớ liên tiếp** đủ chứa 5 số nguyên, và `a[i]` chính là phần tử ở vị trí thứ `i` trong khối đó.

6.1.2 Tính chất và hệ quả

- **Truy cập ngẫu nhiên O(1):** Có thể đọc/ghi `a[i]` ngay lập tức nhờ địa chỉ phần tử được tính theo công thức:

$$\texttt{address(a[i])} = \texttt{address(a[0])} + i \cdot \texttt{sizeof(type)}$$

- **Chỉ số bắt đầu từ 0:** Nếu mảng có n phần tử thì chỉ số hợp lệ là $\{0, 1, \ldots, n-1\}$.

- **Kích thước cố định:** Với mảng kiểu C, số phần tử phải được biết tại thời điểm biên dịch (hoặc phụ thuộc trình biên dịch nếu dùng VLA).
- **Nguy cơ truy cập ngoài biên:** C++ không tự chặn `a[-1]` hay `a[n]` ⇒ dễ gây sai kết quả hoặc lỗi bộ nhớ.

6.1.3 Khai báo và khởi tạo mảng

Cú pháp:

```
kieu_du_lieu ten_mang[so_luong_phan_tu];
```

Ví dụ khai báo:

```
int a[100];           // up to 100 int elements
double b[200];        // up to 200 double elements
char s[20000];        // character array
pair<int,int> p[10]; // array of pairs
```

Khởi tạo khi khai báo:

```
int b[6] = {2, 8, 0, 4, 2, 5};
char arr[6] = {'2', '8', 't', 'e', 'c', 'h'};
```

Lưu ý quan trọng về giá trị ban đầu:

- Nếu bạn khai báo `int a[100];` (không khởi tạo), các phần tử thường là **giá trị rác** (không xác định).
- Nếu bạn khởi tạo thiếu phần tử, các phần tử còn lại sẽ được gán **0**:

```
int b[10] = {2, 8, 0}; // b[0..2] as listed, b[3..9] = 0
```

6.1.4 Thao tác cơ bản trên mảng

1) Truy cập phần tử theo chỉ số

- Cú pháp: `a[i]`
- Ví dụ: Với `b = {2,8,0,4,2,5}` thì `b[3] = 4`.

2) Duyệt mảng thuận và ngược

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    int n = 10;
    int a[10] = {3, 2, 1, 4, 5, 8, 9, 7, 6, 10};

    cout << "Duyet␣thuan␣:␣";
    for(int i = 0; i < n; i++){
        cout << a[i] << "␣";
    }

    cout << "\nDuyet␣nguoc␣:␣";
    for(int i = n - 1; i >= 0; i--){
        cout << a[i] << "␣";
    }

    char arr[6] = {'2', '8', 't', 'e', 'c', 'h'};
    cout << "\nMang␣char␣:␣";
```

```
    for(int i = 0; i < 6; i++){
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output

```
Duyet thuan : 3 2 1 4 5 8 9 7 6 10
Duyet nguoc : 10 6 7 9 8 5 4 1 2 3
Mang char : 2 8 t e c h
```

3) Thay đổi giá trị phần tử

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    int b[6] = {1, 2, 3, 4, 5, 6};

    b[0] = 28;
    b[1] = 100;
    b[2] = 14;
    b[3] += 10;
    b[4] *= 2;
    b[5] /= 2;

    cout << "Mang sau khi thay doi : ";
    for(int i = 0; i < 6; i++){
        cout << b[i] << " ";
    }
    return 0;
}
```

Output

```
Mang sau khi thay doi : 28 100 14 14 10 3
```

4) Nhập mảng từ bàn phím

Cách 1: mảng kích thước cố định

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    int n;
    int a[1000]; // up to 1000 elements

    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }

    for(int i = 0; i < n; i++){
        cout << a[i] << " ";
    }
    return 0;
}
```

Cách 2: mảng theo n (không chuẩn C++ ở nhiều compiler)

- **Lưu ý:** C++ chuẩn **không hỗ trợ** `int a[n];` khi `n` không phải hằng số; một số trình biên dịch cho phép như một mở rộng (VLA).
- Trong thực hành hiện đại, nếu cần kích thước theo `n` thì dùng `vector`.

6.1.5 Mảng làm tham số cho hàm

Bản chất: mảng suy biến thành con trỏ khi truyền vào hàm

Khi truyền mảng vào hàm, tham số `int a[]` thực chất được hiểu như `int* a`. Do đó, thay đổi `a[i]` trong hàm sẽ tác động trực tiếp lên mảng gốc.

Ví dụ: hàm nhập và in mảng

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

void nhap(int a[], int n){
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
}

void in(int a[], int n){
    for(int i = 0; i < n; i++){
        cout << a[i] << "␣";
    }
}

signed main(){
    int n, a[1000];
    cin >> n;
    nhap(a, n);
    in(a, n);
    return 0;
}
```

Ví dụ: thay đổi mảng trong hàm

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

void change(int a[], int n){
    for(int i = 0; i < n; i++){
        a[i] = 28;
    }
}

signed main(){
    int b[6] = {1, 2, 3, 4, 5, 6};
    change(b, 6);

    for(int i = 0; i < 6; i++){
        cout << b[i] << "␣";
    }
    return 0;
}
```

Output

```
28 28 28 28 28 28
```

6.1.6 Các chú ý quan trọng khi dùng mảng

- **Chú ý 1 (ngoài biên):** C++ không tự báo lỗi khi truy cập ngoài biên ⇒ cần kỷ luật chỉ số.
- **Chú ý 2 (giới hạn bộ nhớ):** số phần tử tối đa phụ thuộc `sizeof(type)` và giới hạn bộ nhớ đề bài.
- **Chú ý 3 (stack overflow):** mảng lớn khai báo trong hàm (đặc biệt `main`) có thể làm tràn stack.

Ví dụ lỗi tràn stack

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    cout << "hello␣stack!\n";
    int a[1000000]; // may overflow the stack depending on the environment
    return 0;
}
```

Cách khắc phục: khai báo toàn cục (vùng nhớ lớn hơn stack)

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

int a[1000000]; // global array (stored outside stack)

signed main(){
    cout << "hello␣stack!\n";
    return 0;
}
```

Output

```
hello stack!
```

6.2 Cấu trúc dữ liệu `vector`

6.2.1 Vì sao cần `vector`?

Mảng kiểu C có ưu điểm truy cập nhanh, đơn giản, nhưng có 2 hạn chế lớn trong thực tế:

- Kích thước cố định, không linh hoạt theo dữ liệu nhập.
- Dễ gây lỗi khi khai báo mảng quá lớn trên stack, hoặc khi `n` thay đổi.

`vector` là mảng động trong thư viện STL, giải quyết các vấn đề trên:

- Có thể thay đổi kích thước (tăng/giảm) khi chạy chương trình.
- Cung cấp nhiều thao tác tiện lợi: `push_back`, `pop_back`, `size`, `clear`, ...
- Vẫn hỗ trợ truy cập theo chỉ số như mảng: `v[i]`.

6.2.2 Khai báo và khởi tạo `vector`

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    vector<int> v;                  // empty vector
    vector<int> a(5);               // 5 elements, default = 0
    vector<int> b(5, 7);            // 5 elements, all = 7
    vector<int> c = {2, 8, 0, 4};   // initializer list

    cout << c.size() << "\n";
    return 0;
}
```

6.2.3 Các thao tác cơ bản với vector

1) Thêm/xóa phần tử cuối

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    vector<int> v;

    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    v.pop_back(); // remove the last element (30)

    for(int x : v) cout << x << "␣";
    return 0;
}
```

Output

```
10 20
```

2) Truy cập và sửa phần tử

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    vector<int> v = {1, 2, 3, 4, 5};

    v[0] = 28;
    v[3] += 10;

    for(int i = 0; i < (int)v.size(); i++){
        cout << v[i] << "␣";
    }
    return 0;
}
```

Output

```
28 2 3 14 5
```

3) Nhập vector theo n (cách chuẩn, không phụ thuộc compiler)

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    int n;
    cin >> n;

    vector<int> a(n);
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }

    for(int x : a) cout << x << "␣";
    return 0;
}
```

6.2.4 So sánh nhanh: mảng C vs `vector`

- **Mảng C:** nhanh, gọn, phù hợp khi biết trước kích thước tối đa và cần tối ưu tuyệt đối.
- **`vector`:** linh hoạt, an toàn hơn trong nhập liệu theo `n`, dùng nhiều trong thực tế và thi đấu.

6.2.5 Vector làm tham số hàm: đúng chuẩn và an toàn

Truyền tham chiếu để tránh copy

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

void fill28(vector<int> &v){
    for(int &x : v) x = 28;
}

signed main(){
    vector<int> v = {1,2,3,4,5};
    fill28(v);

    for(int x : v) cout << x << "␣";
    return 0;
}
```

Output

```
28 28 28 28 28
```

Chỉ đọc dữ liệu: dùng `const vector&`

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

int sumVector(const vector<int> &v){
    int s = 0;
    for(int x : v) s += x;
    return s;
}

```

```
12 signed main(){
13     vector<int> v = {2, 8, 0, 4, 2, 5};
14     cout << sumVector(v);
15     return 0;
16 }
```

Output

```
21
```

6.3 Tổng kết chương

- Mảng một chiều là khối ô nhớ liên tiếp, truy cập theo chỉ số, tốc độ nhanh nhưng kích thước cố định.
- `vector` là mảng động của STL, linh hoạt, thuận lợi nhập theo `n`, thao tác đa dạng.
- Khi truyền dữ liệu vào hàm:
  - Mảng C thường được hiểu như con trỏ ⇒ sửa trong hàm sẽ ảnh hưởng mảng gốc.
  - Với `vector`, nên truyền `vector<T>&` để sửa và `const vector<T>&` để chỉ đọc.

CHƯƠNG 7

GIỚI THIỆU VỀ LẬP TRÌNH

CHƯƠNG 8

GIỚI THIỆU VỀ LẬP TRÌNH

CHƯƠNG 9

GIỚI THIỆU VỀ LẬP TRÌNH

CHƯƠNG 10

GIỚI THIỆU VỀ LẬP TRÌNH