Introduction

ใน competitive programming นั้นจะเป็นการรวมเอาเนื้อหา 2 หัวข้อเข้าไว้ด้วยกัน ได้แก่ design of algorithms และ implementation of algorithms นั่นคือออกแบบอย่างไรและเขียนโปรแกรมอย่างไร

สำหรับ design of algorithms นั้น ประกอบด้วย problem solving และ mathematical thinking ทักษะในการวิเคราะห์ปัญหาและแก้ปัญหาอย่างสร้างสรรค์นั้นมีความจำเป็น ทั้งนี้ algorithm สำหรับแก้ปัญหานั้นจะต้องมีความถูกต้องและมีประสิทธิภาพ ซึ่งหัวใจหลักในการออกแบบการแก้ปัญหาคือการสร้าง algorithm ที่มีประสิทธิภาพ

ความรู้ทางทฤษฎีด้านการออกแบบ algorithm นั้นมีความสำคัญสำหรับ competitive programmers โดยทั่วไปแล้วคำตอบของปัญหาคือการรวมกันของเทคนิคที่รู้จัก นำมารวมกับมุมมองใหม่ๆ ทั้งนี้เทคนิคที่เราพบเจอใน **competitive programming** ส่วนใหญ่นั้นมาจากพื้นฐานของงานวิจัยทาง ด้าน algorithms

สำหรับ implementation of algorithms นั้นต้องการทักษะการโปรแกรมที่ดี ใน **competitive programming** นั้น คำตอบจะถูกให้คะแนนโดย grader ด้วยชุดของ test cases ดังนั้นมันไม่เพียงพอว่าแนวคิดการออกแบบของเราถูกต้อง แต่การ implement ก็ต้องถูกด้วย

Good coding style ในการแข่งกันคือการเขียนโปรแกรมอย่างตรงไปตรงมาและชัดเจน โปรแกรมควรถูกเขียนอย่างรวดเร็ว เพราะว่ามีเวลาจำกัด ซึ่งจะไม่เหมือนกับใน software engineering โดยทั่วไป โปรแกรมควรจะสั้น และมันไม่จำเป็นต้องการการ maintain หลังแข่งเสร็จ

Programming Language

- ในปัจจุบัน ภาษาโปรแกรมที่ได้รับความนิยมในการแข่งขันคือ C++, Python และ Java ตามลำดับ จากสถิติของ Google Code Jam 2017 https://www.go-hero.net/jam/17/languages ทั้งนี้ยังมีผู้แข่งขันบางคนที่ใช้ภาษาโปรแกรมนอกเหนือจากนี้

- หลายคนคิดว่าภาษา C++ เป็นตัวเลือกที่ดี สำหรับ competitive programmer และ ส่วนใหญ่ C++ นั้นในการแข่งขันหลายแห่งมีให้ใช้ ข้อดีของ C++ คือเป็นภาษาที่มีประสิทธิภาพและมี standard library ที่มีโครงสร้างข้อมูลและ algorithms ให้ใช้

Language statistics

Number of contestants using specific language.

Language	*Remaining*	Qualification Round	Round 1A	Round 1B	Round 1C	Round 2
C	13	888	88	160	29	17
C#	22	806	111	251	90	39
C++	812	12047	2982	4266	2276	1935
D	5	20	7	10	4	4
F#	1	14	3	4	3	2
Go	5	138	30	37	19	9
Groovy	2	13		3	3	2
Haskell	5	157	17	43	21	8
Java	87	5117	736	1340	449	183
JavaScript	6	259	26	61	20	2
Julia	1	12	1	2	5	2
Kotlin	4	42	8	19	10	9
Lua	2	17	2	6	3	1
OCaml	4	22	6	14	8	2
Pascal	5	25	4	12	3	3
Perl	1	53	8	17	9	
PHP	5	154	15	42	14	
Python	155	5900	894	1723	801	287
Racket	2	9	1	2	1	
Ruby	10	253	35	89	38	13
Rust	10	59	4	10	5	4
Scala	5	93	21	36	15	11
Shell	3	16				
Swift	2	45	2	9	2	1
TypeScript	1	11				
Visual Basic	2	21	3	5	2	1

- มี paper ที่เปรียบเทียบภาษาโปรแกรมในการแข่ง Google Code Jam โดย Back และ Westman

  http://publications.lib.chalmers.se/records/fulltext/250672/250672.pdf
- พวกเขาได้เปรียบเทียบภาษา C, C#, C++, Java และ Python หลายอย่าง เขาบอกว่าจะให้ฟันธงเลยว่าอันไหนดีสุดเป็นเรื่องยากมันขึ้นกับคุณสมบัติที่ตรวจสอบ แต่พวกเขาสรุปว่าถ้าเลือกเวลาและขนาดในการประมวลผล C, C++ และ Python เป็นตัวเลือกที่ดี

Google Code Jam [edit]

Tournament	Finals location	Competitors	1st place	2nd place	3rd place
2018	Toronto, Canada	?	Gennady Korotkevich	Kamil Debowski	Makoto Soejima
2017	Dublin, Ireland[5]	25,289	Gennady Korotkevich	Konstantin Semenov	Vladislav Epifanov
2016	New York City, New York, United States	27,170	Gennady Korotkevich[6]	Kevin Atienza	Egor Kulikov
2015	Seattle, Washington, United States	23,296	Gennady Korotkevich	Makoto Soejima	Bruce Merry
2014	Los Angeles, United States[7]	25,462	Gennady Korotkevich	Evgeny Kapun	Yuzhou Gu
2013	London, United Kingdom	21,273	Ivan Metelsky[8]	Vasil Bileckiy	Vladislav Isenbaev
2012	New York City, United States	20,613	Jakub Pachocki	Neal Wu	Michal Forišek
2011	Tokyo, Japan	14,397	Makoto Soejima	Ivan Metelsky	Jakub Pachocki
2010	Dublin, Ireland	12,092	Egor Kulikov	Erik-Jan Krijgsman	Sergey Kopeliovich
2009	Mountain View, California, United States	8,605[9]	Tiancheng Lou	Zichao Qi	Yoichi Iwata
2008	Mountain View, California, United States[10]	7,154	Tiancheng Lou	Zeyuan Zhu	Bruce Merry
2006	New York City, United States	?	Petr Mitrichev	Ying Wang	Andrey Stankevich
2005	Mountain View, California, United States	?	Marek Cygan[11]	Erik-Jan Krijgsman	Petr Mitrichev
2004	Mountain View, California, United States	?	Sergio Sancho	Po Ruh Loh	Reid Barton
2003	Mountain View, California, United States	?	Jimmy Mårdell	Christopher Hendrie	Eugene Vasilchenko

Gennady Korotkevich

Gennady Korotkevich in 2017.

Born	25 September 1994 (age 24) Gomel, Belarus
Residence	Saint Petersburg, Russia
Other names	"tourist" (handle); Gena (diminutive)
Citizenship	Belarusian
Education	ITMO University
Years active	2005–
Known for	Programming prodigy; highly ranked sport programmer since a very young age

- อย่างไรก็ตาม หากเชี่ยวชาญหลายๆ ภาษาและเข้าใจถึงความแข็งแกร่งของภาษานั้นๆ จะดีมาก ตัวอย่างเช่นหากเราเจอปัญหาที่ต้องการใช้ interger ที่มีขนาดใหญ่มากๆ Python สามารถเป็นตัวเลือกที่ดีได้เพราะว่ามันมี built-in operation สำหรับจัดการการคำนวณ integer ที่มีขนาดใหญ่
- ทั้งนี้ปัญหาส่วนใหญ่ในการแข่งขันนั้นจะถูกออกมาให้การใช้ภาษาโปรแกรมบางชนิดไม่ได้เปรียบเสียเปรียบ

- ในรายวิชานี้เราจะโฟกัสส่วนใหญ่ที่ C++

C++ code template

โครงของ code ภาษา C++ สำหรับ competitive programming เริ่มต้นจะหน้าตาประมาณนี้

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    // solution comes here
    return 0;
}
```

- #include ในบรรทัดแรกเป็น feature ของ g++ compiler ที่ทำรวมเอา standard library เช่น iostream vector และ algorithm มาให้ (ข้อเสียคือ มันจะ include ไฟล์มาหลายอัน อาจจะทำให้ compile time มากได้ ถ้า sensitive เรื่องเวลามากๆ ก็ควร include เฉพาะอันที่ใช้)

- using เป็นการประกาศว่าคลาสและฟังก์ชันต่างๆ ของ standard library สามารถถูกใช้ได้โดยตรงจาก code ซึ่งหากไม่มีบรรทัด using เราจะต้องเขียน code ว่า std::cout แทนที่จะเขียนสั้นๆ ว่า cout

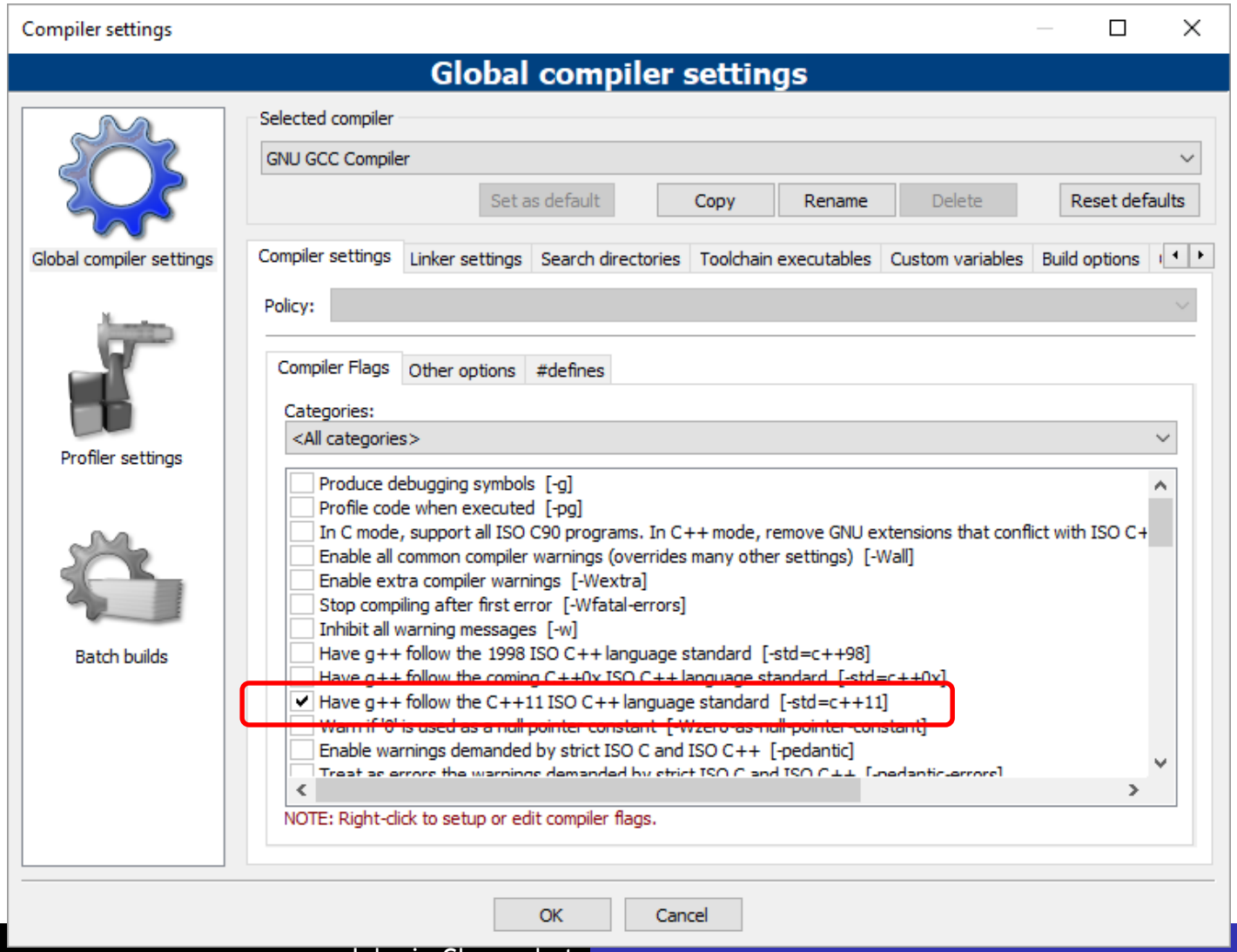
เราสามารถ compile ได้โดยใช้ คำสั่ง

```
g++ –std=c++11 –02 –Wall test.cpp –o test
```

- คำสั่งนี้จะสร้าง binary file ชื่อ test จาก source code test.cpp
- ใช้ compiler C++11 (–std=c++11)
- optimize code (–02)
- และแสดงแจ้งเตือนทุกอย่าง (–Wall)

สำหรับ codeblock จะ set C++11

- เลือกเมนู Settings –> Compiler –> เลือก [–std=c++11]



Input/Output

- ในการแข่งขันส่วนใหญ่นั้น standard streams ถูกใช้ในการอ่านข้อมูลเข้าและเขียนคำตอบออก ใน C++ นั้น standard streams ได้แก่ cin สำหรับการนำข้อมูลเข้าและ cout ในการส่งข้อมูลออก

- เพิ่มเติมหากเป็นใน C ฟังก์ชัน scanf และ printf จะถูกใช้(ทำงานเร็วกว่า cin cout)

- ข้อมูลเข้าของโปรแกรมโดยทั่วไปจะประกอบด้วยตัวเลขและข้อความซึ่งจะแยกกันด้วยช่องว่างและการขึ้นบรรทัดใหม่

- ข้อมูลเข้าสามารถถูกอ่านจาก cin stream ดังตัวอย่างนี้

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

- code ตัวอย่างนี้ทำงานได้โดยสมมติว่าจะมีอย่างน้อยหนึ่ง space หรือ ขึ้นบรรทัดใหม่อยู่ระหว่างข้อมูลเข้าแต่ละตัว ตัวอย่างเช่น

```
123 456 monkey
```

```
123 45
monkey
```

cout stream ถูกใช้ดังตัวอย่างนี้

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

อย่างไรก็ตาม input และ output บางครั้งเป็นจุดที่ทำให้เกิดคอขวดในโปรแกรมคือทำให้โปรแกรมช้า code ต่อไปนี้เมื่อเขียนไว้ในส่วนต้นของโปรแกรมจะช่วยให้ input output มีประสิทธิภาพมากขึ้น

```
ios::sync_with_stdio(0);
cin.tie(0);
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    // solution comes here
    return 0;
}
```

Note คำสั่ง '\n' ทำงานเร็วกว่าคำสั่ง endl เนื่องจากว่า endl จะไป flushก่อน

นอกจากนี้ scanf และ printf ซึ่งเป็นฟังก์ชันในภาษา C เป็นทางเลือกในการรับข้อมูล ทั้งนี้ scanf และ printf ทำงานเร็วกว่า cout และ cin แต่ใช้งานยากกว่าเล็กน้อยเนื่องจากต้องระบุชนิดข้อมูล

ตัวอย่างการรับข้อมูล

```
int a, b;
scanf("%d %d", &a, &b);
```

ตัวอย่างการแสดงผล

```
int a = 123, b = 456;
printf("%d %d\n",a,b);
```

บางครั้งโปรแกรมต้องรับข้อมูลทั้งบรรทัด ซึ่งอาจจะมี space ในนั้น วิธีการจัดการคือใช้ฟังก์ชัน getline

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

- หากรับข้อมูลแบบบรรทัด แต่ว่ามีหลายตัว แต่ละตัวคั่นด้วย space จะใช้ istringstream มาช่วย

```
string s;
int a[99];
getline(cin,s);
istringstream is(s);
N = 0;
while(is>>a[N]) ++N;
```

- หรือถ้าจะประยุกต์ใช้กับ vector

```
string s;
vector<int> a;
getline(cin,s);
istringstream is(s);
int temp;
while(is>>temp) a.push_back(temp);
```

- ข้อควรระวังกับการใช้ getline(cin, var) เรียกว่าปัญหาจะดีกว่า
- ให้ลอง code ต่อไปนี้

```
#include <iostream>
using namespace std;
int main()
{
    string name;
    int age;
    cout<<"Enter your age";
    cin>>age;
    cout<<"Enter your full name";
    getline(cin, name);
    cout<<name<<", you are " <<age<<endl;
    return 0;
}
```

- เนื่องจากว่า getline ไม่ละ whitespace ด้านหน้า เราจะต้องระวังเมื่อใช้ getline ต่อจาก cin

- ปัญหาคือ cin>> จะทิ้ง newline character (\n) ไว้ใน iostream ถ้า getline โดนเรียกถัดจาก cin>> getline ก็จะเห็น \n เป็น whitespace ด้านหน้า มันก็จะคิดว่าจบและหยุดการอ่านต่อไป

- แถบข้อมูลคือ iostream
- คำสั่ง cin จะใช้ 5 และทิ้ง \n ใน stream เป็นขยะ โดยคำสั่ง cin จะไม่อ่าน \n และ cin จะละ \n เมื่ออ่านข้อมูล
- แต่ getline อ่านและเก็บ \n (ตัวด้านซ้าย) มันก็คิดว่าจบการอ่านข้อมูล

- วิธีแก้
- ทางที่ 1
  - หลีกเลี้ยงการใช้ getline ต่อ cin>>
  - วิธีนี้ มีข้อจำกัดเยอะ อย่างเช่นถ้าเราใช้ loop ก็ยากละ
- ทางที่ 2
  - ก็ใช้ getline ในการเก็บข้อมูลทิ้งเลย โดยใช้ตัวแปร temp หรือเรียกว่า dummy variable มาเก็บข้อมูลแต่ไม่ได้ใช้ไว้
  - ใช้ cin.ignore()

ถ้าเราไม่รู้จำนวนข้อมูลว่ามีกี่ตัว เราสามารถจัดการได้โดยการใช้ loop

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

โดย loop นี้จะอ่านข้อมูลมาทีละตัวจาก input จนกระทั่งไม่มีข้อมูลจาก input

- ในบางการแข่งขัน มีการใช้ file สำหรับ input และ output

ทางแก้ที่ง่ายในการเขียน code เหมือนปกติ เราเพียงเพิ่ม code ด้านล่าง

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

หลังจากนั้นโปรแกรมจะอ่านข้อมูลเข้าจากไฟล์ "input.txt" และเขียนผลลัพธ์ลงไฟล์ "output.txt" ซึ่งเราก็ใช้ cin cout ตามปกติ

การทำงานกับตัวเลข

- **เลขจำนวนเต็ม**

  เลขจำนวนเต็มที่ใช้มากที่สุดใน competitive programming คือ int ซึ่งคือชนิดข้อมูล 32-bit ที่มีค่าอยู่ในช่วง -2^{31} ... $2^{31}-1$ หรือ ประมาณ -2×10^9 ... 2×10^9

- ถ้าชนิดข้อมูล int ไม่พอ ชนิดข้อมูลแบบ 64-bit เราก็จะใช้ long long ที่มีค่าอยู่ในช่วง -2^{63} ... $2^{63}-1$ หรือประมาณ -9×10^{18} ... 9×10^{18}

- ตัวอย่างการใช้ long long

```
long long x = 123456789123456789LL;
```

ส่วนท้าย LL บ่งบอกว่ามีชนิดเป็น long long.
จุดที่ผิดเมื่อใช้ long long คือชนิดข้อมูล int ถูกใช้ใน code ตัวอย่างเช่น

```
int a = 123456789;
long long b = a*a;
cout << b << "\n"; // -1757895751
```

- แม้ว่าตัวแปร b จะมีชนิดเป็น long long แต่จำนวนทั้งสองตัวใน expression a*a มีชนิดเป็น int และผลลัพธ์ก็จะเป็น int ทำให้ตัวแปร b เก็บค่าที่ผิด
- ปัญหานี้สามารถแก้ไขได้โดยการเปลี่ยนชนิดของ a ให้เป็น long long หรือเปลี่ยน expression ให้เป็น (long long)a*a.
- โดยทั่วไปปัญหาแข่งขันจะถูกกำหนดให้ long long เพียงพอ
- หากต้องการจำนวนเต็มที่มากกว่านี้อาจจะต้องดู BigInteger

Modular arithmetic

- เราจะแทนด้วย x mod m ว่าเป็นเศษ (remainder) เมื่อ x ถูกหารด้วย m ตัวอย่างเช่น

  17 mod 5 = 2 เพราะว่า 17 = 3x5+2.

- บางครั้ง คำตอบของปัญหาเป็นตัวเลขที่มีขนาดใหญ่มากแต่เราแสดงคำตอบเพียง "modulo *m*" ก็เพียงพอ นั่นคือ เศษเมื่อคำตอบถูกหารด้วย m ตัวอย่างเช่น "modulo $10^9 + 7$"

- แนวคิดคือแม้ว่าคำตอบจริงจะมีค่ามาก มันเพียงพอที่จะใช้ชนิดข้อมูลเป็น int และ long long

- คุณสมบัติที่สำคัญของเศษ คือ ในการบวกลบและคูณ เศษสามารถถูกนำออกมากก่อนดำเนินการ
- (a+b) mod m = (a mod m + b mod m)mod m
- (a–b) mod m = (a mod m – b mod m)mod m
- (a*b) mod m = (a mod m * b mod m)mod m

ดังนั้นเราสามารถเก็บเศษหลังจากทุกการดำเนินการและจำนวนนั้นจะไม่ใหญ่เกินไป

- ตัวอย่างเช่น การคำนวณ n! modulo m

```
long long x = 1;
for(int i=2; i<=n; i++){
    x = (x*i) % m;
}
cout << x%m << '\n';
```

- โดยทั่วไปแล้ว เราต้องการเศษในช่วง 0 ถึง m–1 อย่างไรก็ตามใน C++ และภาษาอื่นๆ เศษของเลขจำนวนลบเป็นได้ทั้ง 0 หรือ เลขลบ วิธีการที่ทำให้มั่นใจว่าจะไม่มีเศษเป็นลบคือเริ่มต้นหาเศษตามปกติจากนั้นบวกด้วย m ถ้าผลลัพธ์เป็นลบ

```
x = x%m;
if(x < 0) x += m;
```

Floating point numbers

- โดยทั่วไปชนิดข้อมูลแบบ floating point ใน competitive programming เป็น 64-bit double และ 80-bit long double ซึ่งมีใน g++ compiler

- ในกรณีส่วนใหญ่แล้ว double ก็เพียงพอ แต่ long double แม่นยำกว่า ทั้งนี้จำนวนจุดทศนิยมของคำตอบจะถูกกำหนดในโจทย์ วิธีการแสดงคำตอบของโจทย์นั้นใช้ฟังก์ชัน printf และกำหนดจำนวนจุดทศนิยมใน formatting string ตัวอย่างเช่น ถ้าต้องการแสดงคำตอบที่มีทศนิยม 9 จุด

```
printf("%.9f\n", x);
```

- ความยากเมื่อใช้เลข floating point คือบางจำนวนไม่สามารถแสดงด้วยเลขจำนวนจริงได้อย่างแม่นยำ และจะมี rounding errors
- ตัวอย่างเช่น

```
double x = 0.3*3+0.1;
printf("%.20f\n", x); // 0.99999999999999988898
```

- เนื่องจาก rounding error ค่าของ x ที่ได้น้อยกว่า 1 ในขณะที่ควรจะตอบ 1

- มันเป็นการ**เสี่ยง**ในการเปรียบเทียบ floating point ด้วย == เพราะว่ามันเป็นไปได้ที่ว่าค่าควรจะเท่ากันแต่มันไม่เป็นเช่นนั้นเพราะว่า precision errors

- ดังนั้นทางที่ดีในการเปรียบเทียบ floating point จะสมมติว่าเลขสองตัวเท่ากันถ้ามันมีผลต่างกันน้อยๆๆ

ในทางปฏิบัติ เราเปรียบเทียบสองจำนวนด้วย $\varepsilon = 10^{-9}$

```
if (abs(a-b) < 1e-9) {
    // a and b are equal
}
```

Shortening code

- การเขียน code ให้สั้นนั้นเหมาะกับ competitive programming เพราะว่าโปรแกรมควรเขียนให้เร็วที่สุดเท่าที่จะทำได้ เนื่องจากเหตุผลนี้ทำให้ competitive programmers มักจะนิยามชื่อชนิดข้อมูลและส่วนอื่นๆ ของ code

- **ชนิดข้อมูล**
  ใช้คำสั่ง typedef ในการตั้งชื่อชนิดข้อมูล ตัวอย่างเช่น long long ยาวเราอาจจะนิยามชื่อสั้นให้เป็น ll

```
typedef long long ll;
```

เริ่มแรกเป็น

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

- หลังจากย่อแล้วเป็น

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

- นอกจากนี้คำสั่ง typedef ยังถูกใช้กับชนิดข้อมูลแบบซับซ้อนด้วย ตัวอย่างเช่น เราจะใช้ vi แทน vector ของ integers และใช้ pi แทน pair ที่เก็บ integer 2 ตัว

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

- **Macros**
  อีกทางหนึ่งในการเขียน code ให้สั้นคือการนิยาม **macros**
- macro หมายถึงข้อความใน code จะถูกเปลี่ยนก่อนการ compile
- ใน C++ นั้น macros ถูกนิยามด้วย #define ตัวอย่างเช่น

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

- เริ่มแรก code เป็น

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

- ย่อได้เป็น

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

- macro สามารถมี parameter หลายตัวได้ ทำให้เราย่อ loop หรือโครงสร้างอื่นๆ ได้ ตัวอย่างเช่น

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

ก่อนย่อเป็น

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

หลังย่อเป็น

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