

Résoudre son premier problème

UVa 11172 — Relational Operators

beOI Training



OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

Introduction

Ce document explique brièvement comment mettre en place l'environnement nécessaire pour programmer et résoudre son premier problème d'algorithmique sur UVa.

Les explications se concentrent sur un environnement Linux et l'utilisation de la ligne de commande. Ces outils sont puissants et correspondent aux outils disponibles en concours, donc il est recommandé de savoir un peu les utiliser.

Pour utiliser Linux, plusieurs options sont possibles, mais la solution la plus simple pour le début est sans doute de l'installer sur une machine virtuelle. Pour des tutoriels, chercher “installer ubuntu sur machine virtuelle” ¹ sur Google.

1. Certaines instructions dans ce document sont spécifiques à Ubuntu donc si vous n'avez pas d'avis, préférez-le à d'autres distributions de Linux.

Table des matières

Prérequis

Programmation

Compilation, test, soumission

Aller plus loin

Environnement de programmation (C++)

Sur Linux :

- ▶ Installer le compilateur g++ : `sudo apt install g++`
- ▶ Utiliser un bon éditeur de texte : l'éditeur par défaut est parfait (gedit, geany, etc.). Plus avancé : vim, emacs.














Sur Windows :

- ▶ Meilleure solution : installer Linux (dual boot ou VM)
- ▶ Sinon on conseille Code::Blocks :
<http://www.codeblocks.org/downloads/binaries>
- ▶ Télécharger le fichier qui ressemble à `codeblocks-16.01mingw-setup.exe`
- ▶ Ne **pas** utiliser d'IDE en ligne ! (mauvaise habitude)

Juges en ligne

- ▶ Juge en ligne = système qui évalue un code en vérifiant son comportement sur beaucoup d'exemples
- ▶ On va utiliser UVA (il faut créer un compte) :
<http://uva.onlinejudge.org>

Registration

Name:	<input type="text" value="Gennady Korotkevich"/>	 
Email:	<input type="text" value="gennady@korotkevi.ch"/>	 
Username:	<input type="text" value="tourist"/>	 
Password:	<input type="password" value="....."/>	 
Verify Password:	<input type="password" value="....."/>	 
Former UVA ID:	<input type="text"/>	
Results email:	<input type="checkbox"/>	
Virtual Judge:	<input type="checkbox"/>	



I'm not a robot



reCAPTCHA

[Privacy](#) - [Terms](#)

Table des matières

Prérequis

Programmation

Compilation, test, soumission

Aller plus loin

Introduction du problème

Énoncé du problème :

<http://uva.onlinejudge.org/external/111/11172.pdf>

Some operators checks about the relationship between two values and these operators are called relational operators.

Given two numerical values your job is just to **find out the relationship between them** that is (i) First one is **greater than** the second (ii) First one is **less than** the second or (iii) First and second one is **equal**.

Lire le contexte en diagonale (un talent à développer).

Format d'input

L'input est donné par l'*entrée standard*, comme si quelqu'un entrait manuellement le fichier ligne par ligne dans la console.

Input

First line of the input file is an integer t ($t < 15$) which denotes how many sets of inputs are there. Each of the next t lines contain two integers a and b ($|a|, |b| < 1000000001$).

Sample input

```
3
10 20
20 10
10 10
```


Input : test cases

L'input est très souvent structuré en plusieurs test cases (instances de test) qui se trouvent dans le même fichier.

Ici, le nombre de test cases t est donné au début.

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

int main() {
    int t; // nombre de test cases
    cin >> t;
    for (int i=0; i<t; i++) {
        // code ici
    }
}
```

Input : données

Il suffit maintenant de lire *a* et *b*.

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

int main() {
    int t; // nombre de test cases
    cin >> t;
    for (int i=0; i<t; i++) {
        int a,b; // données
        cin >> a >> b;
        // calculer la réponse
    }
}
```

Note : cin permet de lire entiers, flottants et strings, qu'ils soient séparés par des espaces ou des retours à la ligne.

Résultat et output

Maintenant qu'on a a et b , il suffit de calculer le résultat.

```
// à l'intérieur de la boucle
if (a < b)
    cout << "<" << endl;
else if (a > b)
    cout << ">" << endl;
else
    cout << "=" << endl;
```

Note : Ce n'est pas un problème de commencer à écrire l'output avant d'avoir lu tout l'input.

Table des matières

Prérequis

Programmation

Compilation, test, soumission

Aller plus loin

Ouvrir la console

Les explications pour la compilation et le test vont se concentrer sur un environnement Linux en console.

Pour ouvrir la console, deux moyens :

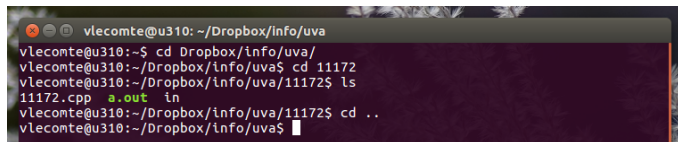
- ▶ L'ouvrir via la liste des applications. Typiquement elle s'appelle "Terminal", "Console" ou "Shell".
- ▶ Depuis l'explorateur de fichier, naviguer vers le dossier puis faire clic droit et "Open in Terminal" ou similaire. Avantage : on est directement dans le bon dossier.

Quand vous ouvrez la console, la ligne qui s'affiche indique le dossier courant : par défaut c'est ~, le dossier "Home", qui contient Documents, Downloads, ...

Commandes de base

Commandes utiles :

- ▶ ls : lister le contenu du dossier courant
- ▶ cd dossier : bouger vers le sous-dossier dossier
- ▶ cd .. : revenir dans le dossier parent



```
vlecomte@u310: ~/Dropbox/info/uva
vlecomte@u310:~$ cd Dropbox/info/uva/
vlecomte@u310:~/Dropbox/info/uva$ cd 11172
vlecomte@u310:~/Dropbox/info/uva/11172$ ls
11172.cpp  a.out  in
vlecomte@u310:~/Dropbox/info/uva/11172$ cd ..
vlecomte@u310:~/Dropbox/info/uva$
```

Raccourcis clavier utiles :

- ▶ Tabulation : permet de compléter le nom d'un fichier/dossier/programme partiellement tapé.
- ▶ Flèches haut/bas : permettent de parcourir l'historique et répéter des commandes déjà exécutées.

Compilation

Commande pour compiler :

```
g++ -std=c++11 -Wall -Wextra -Wshadow -O2 a.cpp
```

- ▶ g++ : nom du compilateur
- ▶ -std=c++11 : version de C++ utilisée
- ▶ -Wall -Wextra -Wshadow : active plein de warnings très utiles pour écrire des programmes sans bugs
- ▶ -O2 : optimise le programme pour l'accélérer
- ▶ a.cpp : code source (il faut être dans le bon dossier !)

Si tout est ok, ça n'affiche rien et crée le programme a.out.
Sinon, lire les messages d'erreur/warning, corriger et réessayer.

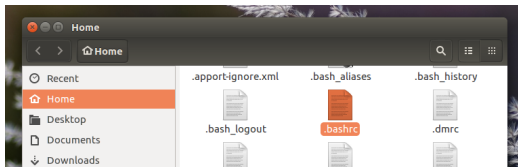
Pour changer le nom a.out, ajouter l'option -o autreNom.

Compilation : pro-tip

Conseil : ajouter la ligne suivante dans à la fin de ~/.bashrc :

```
alias g="g++ -std=c++11 -Wall -Wextra -Wshadow -O2"
```

Le nom .bashrc commence par un point donc est caché par défaut. Pour l'afficher dans un explorateur de fichier, utiliser Ctrl+H. Il s'ouvre comme un simple fichier texte.



Après redémarrage de la console il suffit de taper g a.cpp pour compiler avec toutes les options.

Test et debugging

Pour lancer le programme, il suffit de taper `./a.out`. Si on donne l'input ligne par ligne, le programme le lira et réagira en conséquence.²

Pour éviter de retaper l'input à chaque fois, on peut le sauver dans un fichier texte (par exemple `11172.in`, le nom est sans importance) et le donner au programme automatiquement avec la commande `./a.out < 11172.in`. Pour des programmes plus gros, il est recommandé de créer ses propres inputs.

De manière similaire, on peut sauver l'output du programme dans un fichier texte avec `./a.out > 11172.out` ou même combiner les deux avec `./a.out < 11172.in > 11172.out`. Le fichier d'output est créé s'il n'existait pas encore.

2. Certains problèmes demandent de lire l'input jusqu'à la "fin du fichier" (EOF). Dans ce cas, on peut signaler la fin avec `Ctrl+D`.

Soumission et verdict

Une fois testé, il est temps de soumettre le programme au juge pour vérifier qu'il est correct et assez rapide. Le plus simple est la page "Quick Submit" dans le menu de gauche d'UVa. Entrer le numéro du problème et choisir le langage C++11.

Ensuite, on peut voir le verdict dans "My submissions" :

- ▶ Accepted : le programme est correct, bravo !
- ▶ Wrong Answer : l'output est incorrect
- ▶ Time Limit Exceeded : le programme est trop lent
- ▶ Runtime Error : le programme a crashé
- ▶ Compilation Error : le programme ne compile pas
- ▶ Presentation Error : le format d'output est incorrect

Table des matières

Prérequis

Programmation

Compilation, test, soumission

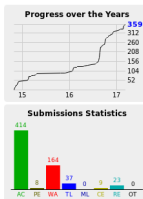
Aller plus loin

Problèmes de sélection beCP

Maintenant que vous avez résolu votre premier problème, vous pouvez continuer sur le 9 autres problèmes pour la sélection aux stages beCP. Plus d'infos : <http://becp.be-oi.be/fr/>

Après ~ 1 jour, vous pourrez utiliser uHunt pour regarder vos statistiques sur UVa : <http://uhunt.felix-halim.net>

Victor Lecomte (vlecomte) statistics



Last Submissions								Show : 5 10 50 100 500 ALL	
Problem		Verdict	Lang	Time	Best	Rank	Submit Time		
11172 - Relational Operator	discuss	Accepted	C++11	0.000	0.000	10422	3 days ago		
1062 - Containers	discuss	Accepted	C++11	0.000	0.000	494	6 days ago		
10935 - Throwing cards away I	discuss	Accepted	C++11	0.000	0.000	1799	6 days ago		
10935 - Throwing cards away I	discuss	Wrong answer	C++11	-	0.000	-	6 days ago		
11988 - Broken Keyboard (a...)	discuss	Accepted	C++11	0.210	0.000	2128	7 days ago		

Solved : 359, Submissions : 655

100 102 103 104 105 108 109 111 115 116 120 122 123 125 146 156 161 183 193 195 200 216 247 259 264 272 294 295 303 314 315 324
325 334 343 344 348 350 357 363 369 374 377 382 389 393 401 403 416 422 429 437 439 441 443 452 454 458 460 488 489 494 514
524 531 543 556 562 563 565 573 579 583 588 590 599 607 610 620 623 624 644 656 662 663 674 705 709 722 737 741 750 752 760
793 796 815 820 825 847 861 893 900 902 920 941 957 976 978 1047 1062 1064 1092 1096 1099 1111 1172 1174 1203 1211 1213 1218
1220 1222 1231 1232 1237 1244 1252 1254 1347 1572 1585 1656 10003 10004 10005 10010 10012 10020 10023 10029 10032 10038
10054 10055 10061 10065 10066 10069 10071 10078 10107 10111 10114 10115 10117 10118 10122 10129 10130 10131 10132 10141
10153 10154 10158 10163 10172 10180 10187 10189 10192 10201 10226 10245 10249 10252 10263 10264 10271 10278 10296 10297
10298 10301 10305 10319 10341 10360 10368 10459 10482 10487 10493 10507 10535 10539 10559 10577 10578 10589 10594 10596
10611 10626 10646 10648 10652 10653 10678 10684 10717 10722 10730 10735 10746 10750 10765 10779 10812 10851 10855 10878

Ressources d'entraînement

Toutes les ressources d'entraînement beCP sont publiées sur notre site : <https://github.com/be-oi/beoi-training>

- ▶ Structuré par unités avec des sujets clairement décrits
- ▶ Slides de tous les cours
- ▶ Exercices liés aux sujets vus

D'autres sites proposent un programme de leçons et exercices assez complet :

- ▶ [France-IOI](#) (français) : site d'entraînement de l'équipe française pour l'IOI.
- ▶ [USACO](#) (anglais) : idem pour les USA.

Plateformes et concours

Il existe de nombreux autres sites avec exercices et concours :

- ▶ **Codeforces** : bons problèmes et concours réguliers
- ▶ **Kattis** : problèmes de qualité, avec indication de niveau³
- ▶ **COCI** : 7 concours par an, très ciblés IOI
- ▶ **USACO** : 4 concours par an, très ciblés IOI, à prendre quand on veut sur un week-end
- ▶ **CodeChef** : trois concours par mois dont un sur dix jours
- ▶ **ACM-ICPC Live Archive** : problèmes d'ACM-ICPC, un concours algorithmique universitaire
- ▶ **SPOJ** : vaste librairie de problèmes

3. Il faut parfois prendre des pincettes pour des problèmes qui n'ont pas été résolus par beaucoup d'utilisateurs.

Livres sur l'algorithmique

Quelques bonnes références sur le competitive programming et l'algorithmique en général :

- ▶ **Competitive Programmer's Handbook** (gratuit) : tout nouveau mais a l'air très bien écrit, et assez complet.
- ▶ **Competitive Programming 3** (payant) : très complet mais parfois ardu à lire. Inclut des problèmes avec chaque sujet, de qualité variable.
- ▶ **Algorithms (Sedgewick, Wayne)** (payant) : bon livre de référence sur l'algorithmique en général.