be-OI 2013 **Finale** 9 mars 2013	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM NOM (MAJUSCULES !): . . . . . . ÉCOLE : . . . . . . LOGIN : MOT DE PASSE : SALLE : CANDIX / DAO	**Réservé**

Olympiade belge d'Informatique (durée : 3h30 maximum)

Ce document est le questionnaire de **la partie machine** de la finale de l'Olympiade belge d'Informatique. Il comporte quatre questions. La première ne rapportera aucun point mais servira d'entrainement pendant le premier **quart d'heure**. Les trois suivantes ne seront distribuées qu'après ce quart d'heure et devront être résolues en **3h30 au maximum**.

Délivrables

1. Votre programme lit les paramètres et données à traiter depuis l'entrée standard et doit écrire son résultat sur la sortie standard. Vous devez rendre votre programme via la système de soumission. Seuls les résultats produits par le code soumis seront utilisés pour calculer votre score finale. Le code lui-même (sa qualité) ne sera pas utilisé par les évaluateurs.
2. L'heure de soumission n'influence pas le score. Cependant, en cas d'égalité sur le score total de la finale, l'heure de soumission du code vous ayant permis d'obtenir le score final sera utilisée afin de départager les égalités et déterminer les positions finales.
3. Vous devez également rendre votre questionnaire, avec le cadre en haut de première page correctement complété.

Notes générales (à lire attentivement avant de répondre à la question)

1. Indiquez votre nom, prénom et école sur la première page. Entourez le nom de la salle dans laquelle vous vous trouvez. Posez votre carte d'étudiant ou carte d'identité sur la table.
2. Installez-vous à la **place** qui vous a été **attribuée** par les organisateurs.
3. Vous ne pouvez avoir que de quoi écrire avec vous, les calculatrices, GSM,... sont **interdits**. Laissez toutes vos affaires à l'endroit indiqué par les surveillants.
4. Vous **ne pouvez** à aucun moment **communiquer** avec qui que ce soit, excepté avec les surveillants ou les organisateurs. Toute question portant sur la compréhension de la question ou liée à des problèmes techniques ne peut être posée qu'aux organisateurs. Toute question logistique peut être posée aux surveillants.
5. Vous **pouvez** utiliser toutes les fonctionnalités de la librairie standard du langage que vous aurez choisi (parmi Java, C, C++, Pascal, Python et PHP) excepté tout ce qui implique une communication avec le monde extérieur hors entrée et sortie standards. En pratique, vous ne pouvez donc pas accéder au réseau, ni lire ou écrire des fichiers sur le disque.
6. Vous pouvez demander des **feuilles de brouillon** aux surveillants.
7. Vous **ne pouvez pas quitter votre place** pendant l'épreuve. Si vous devez absolument vous rendre aux toilettes, faites-en la demande à un surveillant. Ce dernier pourra décider d'accepter ou de refuser votre requête selon qu'il soit possible, ou pas, de vous accompagner tout en assurant la surveillance de l'épreuve.

Instructions pratiques

Ces instructions détaillent comment travailler sur votre programme et ensuite soumettre ce que vous avez écrit sur le serveur officiel. Nous vous conseillons de lire ceci tout en l'appliquant à la *Tâche 0*.

Se connecter sur votre machine et sur le serveur

- Initialement, votre ordinateur affiche une fenêtre d'identification. Choisissez la langue (Français) et l'utilisateur *BeOI user* et cliquez *log-in*.
- Accédez au serveur en double-cliquant sur l'icône *BeOI server* sur le bureau. La page d'accueil du serveur s'affiche. Choisissez votre langue (Français). Sur la page suivante, indiquez le login et le mot de passe qui vous ont été attribués et cliquez sur *Valider*. Votre page principale s'affichera.
- Votre page principale comporte la liste de tâches correspondant à ce questionnaire. Cliquez par exemple sur la tâche 0. La page de cette tâche s'affichera.
- Les pages de tâches sont organisées en deux colonnes. A gauche, les squelettes de programme pour chaque langage et les résultats (initialement vides) de vos soumissions précédentes. A droite, une zone de texte que vous utiliserez pour soumettre vos programmes au serveur.

Charger et modifier les squelettes de programme

- Vous devez d'abord charger les squelettes de programme à partir du serveur. Rendez-vous sur votre page principale. En haut de la page, cliquez sur le lien *Télécharger les squelettes*. Un fichier d'archive `skeleton.tgz` est chargé. Ouvrez-le avec le *Gestionnaire d'archives* (c'est le choix offert par défaut) et extrayez le contenu sur votre bureau (*Archive* > *Extraire* > *Desktop*).
- Sur le bureau, vous trouvez les dossiers que vous venez d'extraire. Il y a un dossier par question (p.ex. `Q0`), chaque question contient un dossier par langage (p.ex. `C`), et chaque langage contient le squelette de programme que vous devez compléter (p.ex. `main.c`), un exemple de données `input.txt` et de résultat attendu correspondant `output-expected.txt`.
- Double-cliquez sur le squelette de programme que vous voulez modifier. Il s'ouvre dans le programme d'édition *gedit*. Vous pouvez maintenant modifier le programme dans *gedit*. D'autres logiciels sont disponibles (voir *Menu des applications* en haut à gauche) mais seul *gedit* a été spécialement configuré pour cette épreuve.
- Si vous désirez récupérer par la suite un seul squelette, ceux-ci sont aussi disponibles sur la page de chaque tâche. Dans la colonne de gauche, section *Codes de départ*, sélectionnez le langage dans lequel vous voulez programmer. Le squelette du programme à compléter pour cette tâche apparaît. Vous pouvez le recopier par copié-collé.

Compiler, exécuter et tester votre programme

- Lorsque vous êtes dans *gedit*, pressez **Ctrl+R** pour tester le programme que vous éditez. Cette commande sauve sur disque, compile, exécute et teste votre programme avec le fichier `input.txt` comme données. Les résultats s'affichent dans une console dans la partie inférieure de *gedit*.
- Si votre programme s'est compilé avec succès, votre programme s'exécute avec comme entrée le fichier `input.txt` présent dans le dossier. Si l'exécution s'est déroulée sans erreur, la sortie de votre programme se trouve dans le fichier `output.txt` et est comparée avec le résultat attendu se situant dans le fichier `output-expected.txt`. Dans tous les cas, si une erreur survient (compilation ou exécution), des informations seront affichées dans la console de *gedit*.
- Vous pouvez modifier `input.txt` pour effectuer d'autres tests, mais dans ce cas vous devez bien entendu adapter le fichier de sortie attendu `output-expected.txt`.

Soumettre votre programme

- La soumission se fait en recopiant votre programme sur le serveur. Dans *gedit*, sélectionnez et copiez tout votre programme (Ctrl-A, Ctrl-C). Sur le serveur, rendez-vous sur la page de la tâche correspondante. Collez votre programme (Ctrl-V) dans la zone de texte dans la colonne de droite, sélectionnez le langage utilisé dans le menu juste au-dessus (attention à ne pas oublier) et cliquez sur *Soumettre*.
- Le serveur compile et évalue votre programme. L'évaluation consiste en une série de tests avec des données différentes. Un temps limite, adapté à chaque langage, est fixé pour chaque test. L'évaluation est interrompue dès qu'un test échoue, soit avec un résultat incorrect soit par manque de temps.
- Les résultats de votre soumission apparaissent en haut de la colonne de droite. Attention, vous devez recharger la page pour mettre à jour ces résultats. Le statut peut être :

  En attente d'évaluation : L'évaluation est encore en cours.

  Erreur de compilation : Votre programme n'a pas pu être compilé.

  Echec : Votre programme a pu être compilé mais un test a échoué suite à un résultat incorrect.

  Temps dépassé : Votre programme a pu être compilé mais un test a échoué suite à un dépassement du temps maximum.

  Réussi : Votre programme a pu être compilé et tous les tests ont réussi.

  Les résultats de chaque test effectué sont donnés juste en-dessous. En cliquant sur le lien *Code*, vous pouvez revoir le code que vous avez soumis. L'historique de vos soumissions pour cette tâche se trouve en bas de la colonne de gauche.
- Seul le résultat de votre programme aux tests automatiques déterminera votre résultat, la qualité de votre code n'entre donc absolument pas en ligne de compte.

Fonctionnement du système de soumission

- Lorsque vous soumettez un nouveau code, celui-ci est mis dans une file d'attente et sera exécuté plus ou moins vite selon le nombre d'autres participants ayant soumis récemment. Il est donc possible qu'en fin de séance, le temps à attendre pour que votre programme soit exécuté atteigne plusieurs minutes.
- Si vous soumettez un nouveau code alors qu'une de vos soumissions précédentes à la même question est encore dans la file, cette dernière sera annulée et la nouvelle soumission remise en fin de file.
- Si à la fin du temps imparti, vous avez encore une soumission en attente, celle-ci sera bien entendu prise en compte. Par contre, vous ne saurez pas combien de points elle vous a rapportés.
- **Seul le meilleur score de chaque tâche sur toutes vos soumissions sera pris en compte pour calculer votre score final à la partie machine.**
- Avant de soumettre, faites bien attention à retirer tout message imprimé à la console que vous auriez ajouté pour *debugguer* votre programme. Ceux-ci étant imprimés à la sortie standard, ils vont rendre le résultat de votre programme incorrect.

Remarques

- Les documentations pour chaque langage se trouvent dans le dossier `Docs` sur le bureau.
- Si vous avez besoin d'aide concernant certains points de cette page, demandez de l'aide à un surveillant.
- Pour tout code non encore soumis sur la plateforme de soumission, il peut être utile d'en faire une copie dans un second fichier pour prévenir les erreurs de manipulation. Ces *« backups »* sont de votre ressort.
- Le code dans *gedit* ne sera jamais évalué, seul le code soumis sur le serveur a de la valeur pour votre score final.

Tâche 0 – (0 pts)

Voici un petit exercice qui vous permettra de tester et de prendre en main votre espace de travail, ainsi que le serveur de soumission. Il s'agit simplement de faire la somme de deux nombres entiers positifs.

Tâche

Écrire un programme qui, étant donné deux entiers positifs, calcule leur somme.

Limites et contraintes

Votre programme ne doit pouvoir gérer que les problèmes se situant dans ces limites. Tous les tests effectués resteront dans ces limites.

- $1 \leq a, b \leq 100$, les nombres à additionner ;

Quoi qu'il arrive, votre programme sera arrêté après **1 seconde** d'exécution, quelque soit le langage utilisé. Votre programme ne peut utiliser plus de **10 Mo** de mémoire.

Entrée

L'entrée donnée à votre programme aura le format suivant :

- La première et unique ligne comporte deux entiers positifs a et b séparés par une espace unique ;
- L'entrée se termine par un saut de ligne.

Sortie

Votre programme écrit en sortie la somme de a et b, suivie d'un saut de ligne.

Exemple

Soit l'entrée suivante fournie à votre programme :

```
10 81
```

La sortie de votre programme doit être :

```
91
```

Tâche 1 – Visite de l'Australie – (30 pts)

Dans le cadre de l'Olympiade Internationale d'Informatique 2013 en Australie, les responsables de la délégation souhaitent organiser quelques visites dans la région de Brisbane durant les jours précédents la compétition. Pour cela, ils préfèrent laisser chacun exprimer ses préférences avec un système de vote original. Chaque membre de la délégation utilise le nombre de points obtenu à la finale nationale pour allouer un poids, selon ses préférence, aux visites qu'il souhaite faire. Par exemple :

Participant A :
- Visite 1 : Parc national Springbrook – 5 pts
- Visite 3 : Baie de Byron – 20 pts
- Visite 5 : Ile de Fraser – 10 pts
- Visite 9 : Plongée à la grande barrière de corail – 37 pts

Participant B :
- Visite 3 : Baie de Byron – 20 pts
- Visite 4 : Mont Coot-tha – 12 pts
- Visite 5 : Ile de Fraser – 10 pts
- Visite 9 : Plongée à la grande barrière de corail – 20 pts

Ces listes sont ensuite combinées deux à deux de la manière suivante afin de déterminer quelles seront les priorités de visite de la délégation. Si une visite n'apparait que dans l'une des deux listes, le score final de cette visite est celui de la liste dans laquelle il apparait. Si une visite apparait dans les deux listes, le score de cette visite est le maximum des deux scores. Par exemple, la combinaison de deux listes ci-dessus sera :

- Visite 1 : Parc national Springbrook – 5 pts
- Visite 3 : Baie de Byron – 20 pts
- Visite 4 : Mont Coot-tha – 12 pts
- Visite 5 : Ile de Fraser – 10 pts
- Visite 9 : Plongée à la grande barrière de corail – 37 pts

En sachant que ces listes sont ordonnées par numéro de visite, on vous demande d'écrire l'algorithme permettant de calculer la liste résultante de la combinaison de deux listes de participants.

Tâche

Vous devez écrire l'algorithme qui permet de combiner deux listes, triées par numéro de visite, selon la règle décrite ci-dessus. Chaque site à visiter sera représenté par un nombre et aura un score (entier positif).

Limites et contraintes

Votre programme ne doit pouvoir gérer que les problèmes se situant dans ces limites. Tous les tests effectués resteront dans ces limites.

- $0 < n, m < 100\,000$, la taille des deux listes.
- $0 < v < 1\,000\,000$, le numéro du site à visiter
- $0 < w < 1\,000\,000$, le poids attribué à une visite

Quoi qu'il arrive, votre programme sera arrêté après **2 sec.** d'exécution, quelque soit le langage utilisé. Votre programme ne peut utiliser plus de **200 Mo** de mémoire.

Entrée

L'entrée donnée à votre programme aura le format suivant :
- Une première ligne contenant n
- n lignes contentant un numéro de site et un poids, séparés par un espace
- Une ligne contenant m
- m lignes contentant un numéro de site et un poids, séparés par un espace
- L'entrée se termine par un saut de ligne.

Sortie

La sortie de votre programme contient un ensemble de lignes contenant deux nombres, séparés par un espace, le numéro de site et le poids de la liste combinée. La liste **doit être ordonnée** par numéro de visite. Le fichier se termine par un saut de ligne.

Exemple

Pour l'entrée suivante

```
4
1 5
3 20
5 10
9 37
4
3 20
4 12
5 10
9 20
```

Votre programme renverra :

```
1 5
3 20
4 12
5 10
9 37
```

Scores

– Tests 1–5 (15 pts) : $0 < n, m < 1\,000$
– Tests 6–10 (15 pts) : $0 < n, m < 100\,000$

Tâche 2 – La grande barrière de corail – (60 pts)

Une plongée sous-marine à la grande barrière de corail ayant élue comme activité préférée, la délégation belge s'y rend pour y voir la plus belle colonie de coraux du monde et plus de 1 500 espèces de poissons et de crustacés. Notre délégation compte bien profiter de cette plongée pour réaliser de magnifiques photos.

La plongée suit un tracé linéaire le long duquel des coraux abritant de nombreux poissons exceptionnels sont observables. Se déplacer d'un site à l'autre prend 5 minutes de nage. A leur arrivée à un nouveau site, nos plongeurs débutants n'ont le temps de prendre qu'une seule photo de chaque poisson avant que ceux-ci ne se cachent dans les coraux pour n'en sortir que 10 minutes plus tard. Ils peuvent ensuite immédiatement décider de continuer leur chemin, de revenir en arrière ou de rester sur place.

Vu la capacité limitée de leurs bouteilles, nos plongeurs ne pourront pas s'attarder indéfiniment sous l'eau et devront remonter à la surface dès que leur bouteille sera vide. Heureusement, le guide expérimenté connait exactement le nombre de poissons à chaque site et nos plongeurs sont des experts en algorithmique. Aidez-les à maximiser le nombre de photos qu'ils pourront faire lors de leur plongée !

Tâche

Etant donnés le temps de plongée et le nombre de poissons à chaque site, on vous demande de donner le nombre maximum de photos qui pourront être prises lors de la plongée en sachant que la plongée commence au premier site. Une marge de sécurité prenant en compte le temps de descente et de remontée est déjà soustraite du temps de plongée. Celui-ci peut donc intégralement être consacré aux photos.

Limites et contraintes

Votre programme ne doit pouvoir gérer que les problèmes se situant dans ces limites. Tous les tests effectués resteront dans ces limites.

- $0 < n < 5\,000\,000$, le nombre de sites.
- $0 < t < 5\,000\,000$, la durée de la plongée en minutes.
- $0 < a_i < 1\,000$, le nombre de poissons au site i.

Quoi qu'il arrive, votre programme sera arrêté après **2 sec.** d'exécution, quelque soit le langage utilisé. Votre programme ne peut utiliser plus de **200 Mo** de mémoire.

Entrée

L'entrée donnée à votre programme aura le format suivant :
- Une première ligne contenant n et t, séparés par un espace.
- n lignes contenant chacune le nombre de poissons a_i pour le i-ième site.
- L'entrée se termine par un saut de ligne.

Sortie

Le nombre maximal de photos de poissons qu'il sera possible de prendre. Le fichier se termine par un saut de ligne.

Exemple

Pour l'entrée suivante

```
5 20
3
4
3
5
1
```

Votre programme renverra :

```
18
```

Explication : Nos plongeurs prennent 3 photos à la 1e position à la minute 0, 4 à la 2e position à la minute 5, 3 à la 3e position à la minute 10, 5 à la 4e position à la minute 15, et reviennent ensuite en arrière pour prendre 3 photos à la 3e position à la minute 20. Ce résultat (18) est le meilleur possible.

Scores

- Tests 1–3 (18 pts) : $0 < n < 200$ et $0 < t < 20$
- Tests 4–6 (21 pts) : $0 < n < 10\,000$ et $0 < t < 10\,000$
- Tests 7–9 (21 pts) : $0 < n < 5\,000\,000$ et $0 < t < 5\,000\,000$

Tâche 3 – La répétition – (60 pts)

Dans le code source de cet exercice se trouve une fonction f. N'essayez pas de comprendre précisément ce que fait cette fonction, vous devez seulement savoir que cette fonction reçoit un nombre entier n comme paramètre, et produit également un nombre entier comme valeur de retour. Cette valeur se trouve toujours dans l'intervalle $[-2^{31}, 2^{31} - 1]$ (c'est-à-dire entre $-2\,147\,483\,648$ et , $2\,147\,483\,647$] inclus), l'intervalle standard pour un nombre entier de 32 bits.

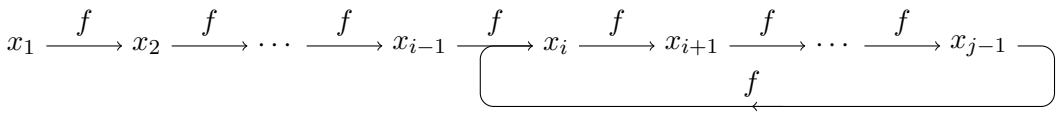
En appliquant la fonction f sur un entier x_1, et en continuant ainsi de suite, on obtient la suite $x_1, x_2, x_3, x_4, \ldots$, telle que $x_2 = f(x_1)$, $x_3 = f(x_2) = f(f(x_1))$, $x_4 = f(x_3) = f(f(f(x_1)))$, ... En d'autres mots, $x_{i+1} = f(x_i)$ pour tout $i \geq 0$.

Par exemple, prenons $x_1 = 1$, et nous obtenons la suite :

$$x_1, \quad x_2, \quad x_3, \quad \ldots \quad = \quad 1, \quad 6, \quad 3, \quad 46, \quad 23, \quad 2646, \quad 1323, \quad 8751642, \quad \ldots$$

(Essayez cet exemple avec un court programme pour vérifier que vous avez bien compris l'explication ci-dessus).

À partir d'un certain point, la suite des nombres va commencer à se répéter (mais cela peut durer longtemps). Il existe donc un index i et un index $j > i$ tels que $x_i = x_j$. Nous illustrons cela par le diagramme ci-dessous :



Nous nommons *période* de la série le nombre de valeurs dans la boucle qui en résulte. En reprenant nos notations, la période vaut donc $j - i$, si i et j ($> i$) sont bien les plus petits indices tels que $x_j = x_i$.

Donnons quelques exemples. Dans la série ci-dessous, la période vaut 3 :

$$10, \quad \underline{100, \quad 33, \quad -2}, \quad 100, \quad 33, \quad -2, \quad 100, \quad 33, \quad -2, \quad 100, \quad 33, \quad -2, \quad 100, \quad 33, \quad -2, \quad \ldots$$

dans cette série, la période est de 5

$$2, \quad 3, \quad \underline{7, \quad 5, \quad -8, \quad 12, \quad -3}, \quad 7, \quad 5, \quad -8, \quad 12, \quad -3, \quad 7, \quad 5, \quad -8, \quad 12, \quad -3, \quad \ldots$$

et ici la période est de 2

$$\underline{-14, \quad 14}, \quad -14, \quad 14, \quad -14, \quad 14, \quad -14, \quad 14, \quad -14, \quad 14, \quad -14, \quad 14, \quad -14, \quad 14, \quad \ldots.$$

Les trois séries d'exemple ci-dessus n'ont rien à voir avec la fonction f qui vous a été donnée. Pour notre fonction f, la période sera beaucoup plus longue, tellement longue qu'on ne peut pas la calculer à la main.

Tâche

Écrivez un programme informatique qui calcule la période de la fonction f pour un nombre initial x_1 donné.

Limites et Contraintes

Pour cette tâche, votre programme n'a droit qu'à une mémoire et un temps d'exécution limités. Vous ne pouvez donc pas utiliser de tableau avec des millions d'éléments, et vous devrez également rechercher une solution efficace, qui n'a pas besoin de plusieurs minutes pour fournir un résultat.

Quoi qu'il arrive, votre programme sera arrêté après **5 sec.** d'exécution (C, Java), **1 min.** d'exécution (Pascal), **10 min.** d'exécution (Python, PHP). Votre programme ne peut utiliser plus de **32 Mo** de mémoire.

Entrée

Un fichier d'une ligne contenant le nombre initial x_1. L'entrée est terminée par un retour à la ligne.

Sortie

Un fichier d'une ligne qui contient la période de la fonction f pour le nombre initial x_1, suivie d'une retour à la ligne.

Exemple

Pour l'entrée suivante :

```
254846291
```

votre programme doit générer la sortie suivante :

```
26
```

Scores

- Test 1 (8 pts) : $i = 1$ (la répétition inclut le nombre de départ), $j < 1000$;
- Test 2 (8 pts) : $i = 1$ (la répétition inclut le nombre de départ) ;
- Test 3 (12 pts) : $i < 1000$ (il y a moins de 1000 étapes avant de trouver un nombre compris dans la répétition) ;
- Test 4 (32 pts) : aucune limite.