Examen de Programmation C/Java

– Examen (**1**) –

Mercredi 9 décembre, 2015
Durée: 3 heures

Résumé

Ce sujet comprend un problème à traiter en C et en Java. À l'issue de l'examen vous devrez envoyer votre travail par e-mail dans une archive contenant le code source des programmes à l'adresse : `emmanuel.fleury@u-bordeaux.fr`.

L'archive devra avoir la forme suivante :

```
NOM_prenom-examen/
    +-- christmas_gifts/
    |     +-- christmas_gifts.c
    |     `-- Makefile
    +-- battleship/
          +--src/
```


1 Programmation C : Christmas Gifts (12 points)

1.1 Mise en place du projet

Questions

1. Sachant qu'il n'y a aucune dépendance à une bibliothèque particulière, réalisez un build-system avec l'outils `make` qui permette de compiler l'exécutable demandé. La structure complète des sources du programme sera la suivante :

```
NOM_prenom-examen/
    +-- christmas_gifts/
          +-- christmas_gifts.c
          `-- Makefile
```

2. Créez le fichier `Makefile` et écrivez les cibles suivantes :
   — `all` : Lance la compilation de l'exécutable `christmas_gifts` ;
   — `christmas_gifts` : Compile le fichier source en un exécutable ;
   — `clean` : Nettoie le répertoire de tous les fichiers superflus et des fichiers crées par la compilation.
3. Au sein de `Makefile` utilisez (et positionnez) correctement les variables classiques, c'est à dire : `CC`, `CFLAGS`, `CPPFLAGS` et `LDFLAGS`. Ainsi que la cible spéciale `.PHONY`.

1.2 Problème principal

L'équipe du père Noël est en plein préparatif pour les fêtes ! Les elfes sont au travail pour préparer les jouets pour les enfants du monde entier. Les elfes responsables de la distribution commencent leur journée en lisant les '`L`' lettres d'enfants au père Noël qu'ils ont à traiter. Ils peuvent ensuite commander des jouets à l'atelier du père Noël parmi une liste de '`N`' types de jouets afin d'essayer de satisfaire les enfants en leur envoyant un (et un seul) des jouets qu'ils ont commandé.

Cependant, chaque type de jouet peut être fabriqué soit avec des matériaux recyclés, soit avec des matériaux neufs (mais c'est plus cher et peu écologique [1]). Et, chaque elfe ne peut commander, par jour, qu'un seul lot de jouets de chaque type qui viendra soit entièrement constitué de matériaux recyclés,

1. Ce sujet est sponsorisé par la COP21 !

soit de matériaux neufs. Il devra donc prendre la décision en début de journée et passera le reste de la journée à empaqueter, affranchir et envoyer les paquets aux enfants. Il a donc '2N' façons de passer sa commande de début de journée et il doit donc déterminer comment passer cette commande.

De plus, chaque lettre d'enfant contient une liste de tous les jouets qui pourraient leur faire envie et qui indique s'ils veulent le jouet en question dans sa version recyclée ou non. Cependant, ce sont des enfants sages et soucieux de la préservation de l'environnement, ils ne choisissent, au plus, qu'un seul jouet neuf dans leur liste (certains, même, ne veulent que du recyclé).

Évidemment, le père Noël, soucieux de l'environnement (et de son porte-monnaie car c'est une icône consumériste), a demandé aux elfes de minimiser le nombre de jouets neufs délivrés. Attention, il arrive de temps en temps que cela ne soit pas possible ! De plus, s'il existe plusieurs solutions, nous nous sommes arrangés pour que la solution qui minimise le nombre de jouets neufs soit unique.

- **Entrée** : La première ligne de l'entrée donne le nombre de cas, `C`. Puis, les `C` cas suivent.
  Pour chacun des cas, la première ligne contient `N`, le nombre de types de jouets à disposition de l'elfe. Puis, à la ligne suivante, vient `L`, le nombre de lettres d'enfants à satisfaire.
  Les `L` lignes suivantes décrivent le contenu des lettres des enfants avec :
  - `S`, le nombre de souhaits de jouets présent sur la liste de l'enfant.
  - Puis, il y a `S` couples `J` et `NE` qui représentent, respectivement, le numéro du jouet dans la liste de l'elfe (`J`) et s'il le veut en neuf ou recyclé (`NE`) (`NE` vaut `1` si l'enfant veut un jouet neuf et `0` sinon).
    Notez bien que :
    - Chaque numéro de jouet `J` n'apparaîtra qu'une fois et une seule sur chaque ligne ;
    - Chaque enfant choisira forcément au moins un type jouet de la liste de l'elfe ($S \geq 1$) ;
    - Chaque ligne aura, au plus, un seul jouet neuf.
- **Sortie** : Pour chaque cas, la sortie sera une ligne contenant `"Case #x: "` (avec `x` qui commence à 1) suivi par :
  - `IMPOSSIBLE` s'il n'existe aucune possibilité de satisfaire les conditions ;
  - Une liste de '`0`' ou de '`1`' séparés par des espaces qui représentent la liste des types de jouets que l'elfe va commander avec `0` s'il commande le lot en neuf et `1` sinon.
- **Limites** : Le nombre de souhaits sur une lettre (`S`) ne dépassera jamais 3000.
  - **Petits ensembles de données** : $C = 100$, $1 \leq N \leq 10$ et $1 \leq L \leq 100$.
  - **Gros ensembles de données** : $C = 5$, $1 \leq N \leq 2000$ et $1 \leq L \leq 2000$.

Fichier d'entrée

```
2
5
3
1 1 1
2 1 0 2 0
1 5 0
1
2
1 1 0
1 1 1
```

Fichier de sortie

```
Case #1: 1 0 0 0 0
Case #2: IMPOSSIBLE
```

Figure 1 – Exemple de fichier d'entrées/sorties.

Dans l'exemple donné, on voit qu'il y a deux cas à traiter, le premier cas dispose d'une liste de 5 types de jouets et de 3 lettres d'enfants à satisfaire. Le premier enfant souhaite uniquement un jouet neuf de type 1. Le deuxième enfant donne le choix entre deux types de jouets : 1 ou 2, tous neufs. Enfin, le troisième enfant veut un jouet de type 5 recyclé. L'elfe peut passer sa commande en demandant des lots recyclés pour tous les types de jouets, sauf le type 1 pour satisfaire le premier enfant.

Pour le second cas, l'elfe a un seul type de jouet à disposition et deux lettres d'enfants, le premier demande le jouet recyclé et le second le jouet neuf. C'est donc impossible.

Enfin, vous trouverez des fichiers d'exemples plus aboutis à l'URL suivante :
`http://www.labri.fr/~fleury/courses/programming/master/exam/`

Questions

1. Commencez par écrire la partie du programme qui ouvre un fichier pour récupérer les données du problème. On suppose que l'interface utilisateur du programme se comportera comme suit :

   ```
   $> ./christmas_gifts
   christmas_gifts: error: No input file given !
   $> ./christmas_gifts extra-small.in
   Case #1: 1 0 0 0 0
   Case #2: IMPOSSIBLE
   ```

   En cas d'absence du fichier, le programme doit terminer en renvoyant une erreur sur `stderr` et avec un code de retour valant `EXIT_FAILURE`.

   Pour le reste du parsing, on supposera que nous avons toujours affaire à des fichiers "parfaits", sans erreur de syntaxe, ni oubli de la part de l'utilisateur. Inutile, donc, de passer du temps à essayer d'être robuste lorsque vous parsez le contenu du fichier. Par contre, on s'attend à ce que vous détectiez l'absence du fichier, ou un problème lors de l'ouverture (en lecture) de celui-ci.

   Il vous faut donc, dans l'ordre, vérifier la présence d'un argument sur la ligne de commande, ouvrir le fichier en lecture (et vérifier qu'il a bien été ouvert), récupérer le nombre de cas, puis faire en sorte de récupérer les éléments de chaque cas.

2. Enfin, lisez les quelques indices algorithmiques ci-dessous et programmez la résolution de ce problème.

   Comme vous l'avez déjà sans doute remarqué, il s'agit d'un problème de satisfiabilité. Chaque lettre d'enfant peut être vu comme une clause contenant des littéraux qui représentent le type de jouet que désire l'enfant et avec, au plus, l'un des littéraux qui est sous forme négative (neuf). La formule globale est la conjonction de toutes ces clauses. Par exemple, le premier cas de l'exemple donné, si on suppose que p_i est le littéral qui représente un jouet de type '`i`', se traduit par :

   $$(\neg p_1) \wedge (p_1 \vee p_2) \wedge (p_5) \tag{1}$$

   Évidemment, le cas général de SAT est NP-complet, mais le fait de ne prendre, au plus, qu'un seul littéral négatif par clause en fait un cas très spécial pour lequel il existe un algorithme linéaire. Nous nous contenterons ici de l'algorithme quadratique (qui est plus simple et suffisant dans notre cas).

   On débute avec tous les types de jouets en 'recyclés' et on considère chaque lettre d'enfant une par une en appliquant les règles suivantes :
   — Si vous trouvez un enfant qui ne désire que des jouets recyclés et que tous ces jouets ont été passés en neufs. Alors, il n'y a pas de solution.
   — Si vous trouvez un enfant qui ne peut être satisfait sauf grâce à un choix neuf, alors basculez le lot en question en neuf et revérifiez si les contraintes sont à présent satisfaites.
   — Si tous les enfants ont été satisfaits, alors vous avez trouvé une solution minimale.

   Notez que lorsqu'on bascule un lot en neuf, c'est que les contraintes nous forcent de le faire. Nous garantissons ainsi d'avoir le nombre minimum de lots neufs.

   Programmez la résolution du problème en utilisant l'algorithme suggéré. La clarté du code, son efficacité ainsi que les commentaires que vous y mettrez seront aussi évalués.

2 Programmation Java (8 points)

Il s'agit d'étendre ici la hiérarchie de petits jeux de devinettes vus en cours, en y incluant un jeu simplifié de bataille navale 1D sur un intervalle $I = [0, n]$, $n \geq 5$. Les bateaux y seront donc de simples sous-intervalles de I. Par exemple :

Indications :

- *Faire en sorte que deux joueurs* automatiques *puissent s'affronter en tentant de "torpiller" à tour de rôle les bateaux de l'autre.*
  Les joueurs possèderont donc chacun une copie de I, et l'on pourra y fixer initialement la position de leurs bateaux respectifs.
  Un tour de jeu (un "torpillage") consistera alors pour un joueur P_1 à choisir un nombre m inclus dans I et à se voir répondre par l'autre joueur P_2 soit par "`miss`*" (aucun des bateaux de P_2 n'a été touché, i.e. m n'est pas inclus dans un des sous-intervalles qui constituent les bateaux de P_2), "*`hit`*" (un des bateaux de P_2 a été touché), "*`sunk`*" (un des bateaux de P_2 a été coulé, i.e. toutes les cases de ce bateau ont été touchées en prenant en compte les tours précédents), ou "*`allsunk`*" (tous les bateaux de P_2 ont été coulés). Le jeu s'arrête lorsqu'un "*`allsunk`*" a été émis.*
- *On se restreindra au cas où chaque joueur dispose exactement de trois bateaux de longueur $n/5$ (où "/" dénote ici la division entière).*
- *Les classes et les interfaces implémentées pour l'exercice seront ajoutées et associées au corrigé de l'exercice 2 de la fiche 2 de TD disponible sur le site du cours.*
- *Vous commenterez vos sources si nécessaire (les commentaires en Java sont encadrés par* `/* ... */`*, ou en préfixant chaque ligne par* `//`*).*
- *Vous rendrez l'ensemble des sources des classes et des interfaces sous forme d'un unique "*`.tgz`*" (ces sources sont situés dans un répertoire de votre "*workspace Eclipse*").*

Questions

1. Implémenter ce jeu complètement, et cela *en appliquant les principes de la programmation objet*, à la manière de ce qui a été vu en cours. Néanmoins, il suffira de n'implémenter qu'un seul type de joueur qui ne fera que tirer aléatoirement des torpilles dans l'intervalle de l'autre joueur. C'est évidemment une simplification, mais elle suffira dans le cadre de cet examen (faire néanmoins en sorte qu'il soit facile d'intégrer *a posteriori* d'autres types de joueurs).
2. Implémenter une classe de test (modifier la classe `GameLaunch` et son `Main`) qui démontrera le bon fonctionnement de votre programme, et cela avec la configuration de bateaux de l'exemple ci-dessus, appliquée aux deux joueurs. Faire en sorte que l'exécution du programme produise une trace des coups de chacun des joueurs. Recopier l'une de ces traces et l'ajouter en commentaire à la fin de votre classe `GameLaunch` avant de rendre votre travail.