EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours de Programmation I, sections MA et PH
Chappelier J.-C.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

NOM : Hanon Ymous
(000000) #0000
Place : 1

Programmation I (SMA/SPH) : SÉRIE NOTÉE

(19 novembre 2015)

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre série annulée dans le cas contraire.

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (10h15 - 12h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
3. Vous avez droit à toute documentation papier.
   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée ; utilisez aussi le verso des feuilles, **MAIS** n'utilisez *que* le verso de la feuille sur laquelle se trouve la question, et non **pas** celui de la feuille précédente !
   Vous pouvez, si nécessaire, joindre des feuilles supplémentaires, mais veillez dans ce cas à *n'utiliser que le papier officiel fourni par les assistants* ; **aucune feuille non officielle ne sera corrigée**.
   N'oubliez pas dans ce cas d'*indiquer votre numéro interne* (0000) sur chacune des feuilles supplémentaires, en haut à droite, ainsi que le numéro de la question traitée. Aucune feuille sans ces informations ne sera corrigée.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. L'examen comporte deux exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 74). Tous les exercices comptent pour la note finale.

NE RIEN ÉCRIRE SUR CETTE PAGE

Question 1 Terminator [sur 15 points]

Le but de cet exercice est d'écrire un programme permettant de simuler l'évolution d'une production de robots. Ces robots pouvant eux-mêmes construire d'autres robots, leur nombre évolue de plus en plus : chaque heure il est multiplié par « un plus le taux de production par heure ».

Au départ, il faut 3 heures pour produire un robot, c'est-à-dire que le taux de production par heure est de 33.3% et que donc le nombre de robots est multiplié chaque heure par 1.333.

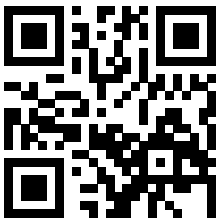
Mais chaque fois qu'au moins 100 robots de plus ont été créés, la capacité de production augmente de 25% (car il y a plus de robots neufs), c'est-à-dire qu'à chaque fois que le nombre de robots augmente d'au moins 100, le taux de production par heure est multiplié par 1.25.

Votre programme devra tout d'abord demander le nombre d'heures de simulation (et le redemander tant qu'il n'est pas strictement positif) puis afficher une estimation (non entière) du nombre de robots après chaque heure d'évolution.

Un exemple de déroulement pourrait être :

```
Combien d'heures (> 0) ? -4
Combien d'heures (> 0) ? 21
     1 : 1.333 robots, taux : 33.3
     2 : 1.77689 robots, taux : 33.3
     3 : 2.36859 robots, taux : 33.3
    [...]
    15 : 74.5508 robots, taux : 33.3
    16 : 99.3762 robots, taux : 33.3
    17 : 132.468 robots, taux : 41.625
    18 : 187.608 robots, taux : 41.625
    19 : 265.701 robots, taux : 52.0312
    20 : 403.948 robots, taux : 65.0391
    21 : 666.672 robots, taux : 81.2988
Nombre de robots après 21 heures : 666.672
```

Ecrivez votre programme ici et au dos (il ne doit pas être plus long que cela) :



Question 2 Places de park [sur 59 points]

On cherche ici à écrire un programme permettant de gérer la répartition de voitures dans des parkings. Au niveau de cet exercice, nous ne vous demandons que six sous-parties du programme complet :

1. les structures de données ;
2. l'affichage d'un parking et ses réservations (ensemble des voitures affectées) ;
3. l'affichage de dates ;
4. le calcul de la place restant dans un parking à un moment donné ;
5. la recherche du meilleur parking ;
6. et enfin deux fonctions outils pour calculer/manipuler les dates et durées de parking.

Pour vous donner une idée, voici typiquement le genre de `main()` que l'on pourrait écrire :

```
vector<Parking> mes_parkings({
{ // premier parking
  "P1", 10, 10,
  { // ses voitures
    {"Volvo"      , 1.75, 1512031040, 1512041620 } ,
    {"Lamborghini", 2.0,  1511231757, 1511210945 } ,
    {"Ferrari"    , 2.0,  1511191015, 1511191200 } ,
    {"Renault"    , 1.5,  1512251200, 1512251700 }
  }
},
{ // second parking
  "P2", 15, 5,
  {
    {"Mercedes"  ,  1.85, 1512031140, 1512051630 } ,
    {"Volkswagen",  1.5,  1510031020, 1510041020 } ,
    {"BMW"       ,  1.8,  1510031040, 1510041620 } ,
    {"Alpha Romeo", 1.75, 1511301135, 1512011510 }
  }
}
});

// Exemples des fonctions demandées :

affiche(mes_parkings[0]);  // === point 2 ===============

affiche_date(1511191045); // === point 3 ===============

// === point 4 ========================================
Voiture peugeot = {"Peugeot",  1.5, 1512031040, 1512041620 };
if (a_de_la_place(mes_parkings[0], peugeot)) {
    cout << "OK" << endl;
} else {
    cout << "pas de place" << endl;
}

/// ... points 5 et 6 au dos .......
```

suite au dos ☞

```
// === point 5 ========================================
size_t park = cherche_meilleur_parking(mes_parkings, peugeot);
if (park < mes_parkings.size()) {
  cout << "Trouvé :" << endl;
  affiche(mes_parkings[park]);
} else {
  cout << "Rien trouvé ! :-(" << endl;
}

// === point 6 ========================================
int fin(1512041610);
int debut(1512031030);
cout << "Temps entre le "; affiche_date(debut);
cout << " et le "; affiche_date(fin);
cout << " : " << calcule_temps(fin, debut, 60) << endl;
```

Question 2.1 Structures de données [sur 5 points]

Dans ce programme, nous aurons besoin de deux sortes de données :

- les « voitures », qui sont caractérisées par : une marque, une largeur (espace que la voiture occupe), la date du début de parking (que l'on représentera par un `int`, voir les explications plus loin) et la date de la fin de parking ;
- le « parking » qui a un nom, une largeur totale (place totale disponible), un prix de location par heure et une liste des voitures qui y sont parquées ou qui ont une réservation pour un parking futur.

Ecrivez ici la définition des *types* de données qui vous semblent appropriés pour représenter les données requises :

Question 2.2 Affichage d'un parking [sur 7 points]

On suppose qu'il existe déjà une fonction `affiche` qui permet d'afficher une voiture.
Ecrivez alors ici la définition d'une fonction `affiche` permettant d'afficher un parking :

Question 2.3 Affichage d'une date [sur 6 points]

Dans ce programme, les dates sont représentées sous forme d'un entier au format « AAMMJJHHMM » où « AA » sont les 2 derniers chiffres de l'année, « MM » les 2 chiffres du mois, « JJ » les 2 chiffres du jour, « HH » les 2 chiffres des heures et « MM » les minutes.

Par exemple, l'entier 1511191045 se décompose en (20)15, 11, 19, 10, 45 et représente donc le 19 novembre 2015 à 10h45.

Le gros avantage de ce format est de pouvoir facilement comparer des dates : la comparaison usuelle d'entiers permet de comparer des dates.

Mais son inconvénient est sa lisibilité. Pour cela, nous vous demandons d'écrire une fonction `affiche_date` qui reçoit un entier au format précédent et écrit (sur `cout`) la date au format usuel : « jour/mois/année `à` heure'`h`'minutes ». Par exemple pour 1511191045, elle écrit « `19/11/2015 à 10h45` ».

Ecrivez ici la définition de la fonction `affiche_date` :

suite au dos ☞

Question 2.4 Trouver de la place [sur 18 points]

On cherche maintenant à savoir si l'on peut placer une voiture dans un parking à une date donnée. Pour cela, on décompose le problème en deux fonctions :

- `place_restante` qui prend un parking et une date en paramètres et retourne la place restante dans ce parking à cette date ;
  pour calculer la place restante, il suffit de soustraire à la place totale du parking la somme des largeurs des voitures dont la date de début de parking est inférieure ou égale à la date donnée et dont la date de fin de parking est strictement supérieure à cette date donnée ;
- `a_de_la_place` qui prend un parking et une voiture en paramètres et retourne vrai ou faux en fonction que la voiture aura la place de se parquer pendant toute la durée désirée (rappel : une voiture contient elle-même ses dates de début et de fin désirées) ;
  il suffit pour cela de vérifier s'il y a déjà assez de place au début de la période de parking voulue ; puis ensuite, pour chaque voiture étant déjà enregistrée dans le parking et dont la date de début de parking est comprise dans la période désirée, s'il reste encore assez de place lorsque cette voiture est là (disons à sa date de début plus 1 minute).

Ecrivez ici les définitions des fonctions `place_restante` et `a_de_la_place` :

Question 2.5 Chercher le meilleur parking [sur 8 points]

On veut maintenant rechercher le meilleur parking pour une voiture donnée, c'est-à-dire parmi un ensemble de parkings, celui qui a de la place pour parquer la voiture et qui a le prix horaire le plus petit.

Ecrivez pour cela une fonctions `cherche_meilleur_parking` qui prend en paramètres un tableau dynamique de parkings et une voiture et retourne la position (dans le tableau) du meilleur parking pouvant recevoir la voiture en question. Cette fonction retournera la taille du tableau de parkings si aucun parking n'est disponible (voir l'exemple de `main()` donné au début) et retournera la position de l'un quelconque des meilleurs parkings en cas d'égalité.

Ecrivez ici la définition de la fonction `cherche_meilleur_parking` :

suite au dos ☞

Question 2.6 Calculs de temps [sur 15 points]

Note : cette partie est plus difficile à expliquer (et à comprendre) qu'à écrire sous forme de code.

Ecrire une fonction `calcule_temps` qui prend en paramètres :
- une date de fin, donnée sous forme d'entier au format précédent (complet ou raccourci ; voir exemples ci-dessous) ;
- une date de début, au même format ;
  on supposera sans le vérifier que cette date est bien au même format et est antérieures à la date de fin ;
- une « précision de mesure » utile pour la conversion comme expliqué ci-dessous et indiquant combien l'unité de temps utilisée est contenue dans l'unité de temps strictement supérieure :
  - 60 pour les minutes, car une heure contient 60 minutes ;
  - 24 pour les heures, car un jour contient 24 heures ;
  - 30 pour les jours, car pour simplifier au niveau de cet examen, on considérera que tous les mois ont 30 jours ;
  - 12 pour les mois, car un an contient 12 mois ;
  - et 0 pour les ans, car on ne considère pas d'unité de temps plus grande que les ans.

La fonction `calcule_temps` retourne sous forme d'entier le nombre d'unités de temps écoulées entre deux dates reçues en paramètres. De plus, elle devra réduire le format des paramètres reçus pour passer à l'unité de temps strictement supérieure.

Par exemple, si la date de fin est `1512041610` (signifiant le `04` décembre (= `12`) `2015` à `16h10`), et la date de départ est `1512031030` (signifiant le `03` décembre (= `12`) `2015` à `10h30`), les dates expriment des minutes et `calcule_temps` devra alors :
- renvoyer `40` (il y a 40 minutes entre `10h30` et `11h10` ; cette valeur 40 résulte du calcul $10 - 30 = -20$ auquel on ajoute 60 puisqu'il est négatif et qu'il y a 60 minutes dans une heure) ;
- modifier le second argument pour qu'il devienne `15120310` (la dernière unité exprimée sont maintenant des heures) ;
- modifier le premier argument pour qu'il devienne `15120415` ; notez bien le `15` à la fin et non plus `16` puisque la différence des minutes était négative (on a entamé l'heure suivante).

Un second appel à `calcule_temps` avec `15120415` (signifiant le `04` décembre (= `12`) `2015` à `15` heures) comme premier argument, `15120310` (signifiant le `03` décembre (= `12`) `2015` à `10` heures), comme second argument et 24 comme troisième argument devra :
- renvoyer `5` (il y a 5 heures entre `10` et `15` heures) ;
- modifier le premier argument pour qu'il devienne `151204` (la dernière unité exprimée sont maintenant des jours) ;
- modifier le second argument pour qu'il devienne `151203`.

Et l'on procéderait ainsi de suite de manière similaire par appels successifs jusqu'aux années (`15` et `15`).

Ecrire enfin une fonction `temps_de_location` qui prend une voiture en paramètre et retourne le nombre d'heures pendant lesquelles la voiture est/sera parquée. Cette fonction devra utiliser la fonction `calcule_temps`.

Ecrivez ici les définitions des fonctions `calcule_temps` et `temps_de_location` :

(place supplémentaire si nécessaire)