ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours de Programmation Orientée Système, sections IN et SC
Chappelier J.-C.

NOM : Hanon Ymous
(000000)
Place : 0

#0000

PROGRAMMATION ORIENTÉE SYSTÈME

Examen

27 mai 2019

INSTRUCTIONS (à lire attentivement)

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

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (9h15 - 11h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
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 ; ne joignez aucune feuille supplémentaire ; **seul ce document sera corrigé**.
   Vous avez si nécessaire de pages blanches supplémentaires en fin de sujet.
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 trois 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 90 points) ; tous les exercices comptent pour la note finale :
   — question 1 : 29 points ;
   — question 2 : 18 points ;
   — question 3 : 43 points.

Question 1 Questions de cours [29 points]

Question 1.1 Petit bout de code manquant [2 points]

Ecrivez la/les ligne(s) de C qui permet(ent) que le code suivant compile :

```
#include <stdio.h>

int main() {
  direction_t d = { 0, 1 };
  printf("%d %d\n", d.dx, d.dy);
  return 0;
}
```

Réponse :

Question 1.2 C quoi ça ? [5 points]

Pour le programme ci-dessous, écrivez ce qu'il affiche, puis décrivez *brièvement* (en *quelques* mots) ce qui se passe à chacune des lignes 12 à 15.

```
 1  #include <stdio.h>
 2  void f(int a, int* b, int c, int* d) {
 3    printf("%d: %d, %d, %d\n",
 4           ++(*d), a, *b, c);
 5  }
 6  int main(void)
 7  {
 8    int  p = 8;
 9    int* q = &p;
10    int  r = 7;
11    int  s = 1;
12    f(p, q, r, &s);
13    *q = 9 ; f(p, q, r, &s);
14     q = &r; f(p, q, r, &s);
15    *q = p ; f(p, q, r, &s);
16    return 0;
17  }
```

Réponse :

Question 1.3 Critique de code [6 points]

La fonction ci-dessous est-elle correcte ? Si non, expliquez (la ou) les erreur(s) et corrigez la/les :

```
char* init(char c, size_t taille) {
  char string[taille];

  for (size_t i = 0; i <= taille; ++i) {
    *((&string)+i) = c;
  }
  return string;
}
```

Réponse :

Question 1.4 Size matters [7 points]

Soit la structure

```
struct abc { int a, b, c; };
```

Le code suivant est-il correct ? Si non, corriger toutes les erreurs (récrivez le code à droite).

Note : la correction d'une erreur qui n'en est pas une sera pénalisée.

```
int reset(struct abc abc) {
  abc.a = 0;
  (*abc)->b = 0;
  &abc->c = 0;
}
```

Supposons qu'un entier soit stocké sur 4 octets, un pointeur sur 8 octets et que le compilateur réserve la stricte place nécessaire pour les variables dans une structure.

Qu'affiche alors le code suivant (répondez à droit du code) :

```
const char* s = "abcde";
struct abc x = { 1, 2, 3 };
printf("%zu, %zu, %zu, %zu, %zu, %zu, %zu\n",
       sizeof(s),
       sizeof(x),  sizeof(struct abc),
       sizeof(&x), sizeof(struct abc *),
       sizeof(x.a),
       sizeof(x.a + 6)
      );
```

suite au dos ☞

Question 1.5 Faire son marché [9 points]

On considère des données au format suivant :

```
Marchandise mon_panier[] = {
    { "pommes",  12.35 },
    { "carottes",  9.70 },
    { "oignons", 11.40 },
    { "" } };
```

où la convention est que la dernière marchandise a simplement un nom vide.

Écrivez une fonction `print_panier()` qui prend un tableau de `Marchandise` respectant la convention précédente (et donc, *pour une fois*, on ne passe pas la taille du tableau en paramètre) et l'affiche comme ceci :

```
On a acheté pour
    12.35 F de pommes
    9.70 F de carottes
    11.40 F de oignons
Total : 33.45 F
```

Si le panier est vide, on affiche simplement « `On n'a rien acheté` ».

Les champs de `Marchandise` se nomment `nom` et `prix`.

Contraintes :

1. Vous devez utiliser *uniquement* l'arithmétique des pointeurs (c.-à-d. ne devez pas utiliser d'index (`size_t`, `int`, ...), en aucune façon).
2. Vous ne pouvez utiliser aucune fonction `str...()` ; donc, pas `strlen()`, ni `strcmp()`, ni rien de la sorte.
3. Vous *pouvez* utiliser `printf("%s"` (et autres).

Réponse :

Suite réponse Question 1.5 :

suite au dos ☞

Question 2 Correction d'erreurs [18 points]

Voici ci-dessous et ci-contre trois (courts) *extraits* d'un programme C qui implémente un interpréteur de commandes sur des chaînes de caractères. L'idée générale (non présente ici) est d'entrer des commandes (comme « *reverse* », « *replace* », etc.) sur des chaînes de caractères.

Ici, nous n'avons qu'une portion du code qui contient :

- deux structures de données : l'une pour représenter une commande de cet interpréteur, l'autre pour représenter l'ensemble des commandes possibles ;
- une fonction `add_command()` permettant d'ajouter une commande à l'interpréteur ;
- une des fonctions pouvant opérer sur les chaînes de caractères et dont le but est d'inverser toute la chaîne (produire la chaîne réécrite à l'envers).

On utiliserait typiquement ces éléments de la façon suivante :

```
add_command(&lang, "reverse", reverse);
```

Vous n'avez rien à supposer ou connaître sur le reste du programme pour répondre à cette question.

Trouvez dans les extraits de code fourni, un maximum d'erreurs possible **et** proposez à chaque fois une correction. Vous pouvez répondre directement sur le code, ainsi qu'à droite ou en dessous de celui-ci.

Attention ! Nous retirons 1 point pour toute indication d'une erreur qui n'en est pas une. (sinon il suffirait de dire que chaque ligne est fausse...)

Notes :

1. Il ne s'agit pas de réécrire tout le code : lorsqu'une erreur potentielle pourrait se corriger de plusieurs façons, choisissez celle qui nécessite le moins de modifications du code fourni.
2. Pour gagner de la place, nous n'avons reproduit qu'une partie du code ; il n'y a pas d'erreur due à du manque de code.

```
 1  typedef char* Operator(const char*);
 2
 3  typedef struct {
 4    const char* name;
 5    Operator op;
 6  } Command;
 7
 8  typedef struct {
 9    size_t size;
10    Command* commands;
11  } Language;
```

```
21  int add_command(Language* l, const char* name, Operator op)
22  {
23    int error = 0;
24    l->commands = realloc(l->commands, ++(l->size) * sizeof(Command));
25    if (l->commands != NULL) {
26      l->commands[l->size].name = name;
27      l->commands[l->size].op   = op;
28    } else {
29      error = 1;
30    }
31    return error;
32  }
```

```
41  char* reverse(const char* s)
42  {
43    const size_t size = sizeof(s);
44    char* new = malloc(size + 1);
45    if (new != NULL) {
46      for(const char* p = s + size; p >= s; --p) {
47        *++new = *p;
48      }
49      new -= size;
50    }
51    return new;
52  }
```

Question 3 Suivi de bugs [43 points]

On s'intéresse ici à un système de « *bugtracking* » permettant aux utilisateurs de soumettre les bugs qu'ils rencontrent et aux développeurs de les gérer. Dans cet exercice, nous allons considérer un modèle simplifié de bugtracking : ce système gère des projets comportant certains bugs, et des programmeurs qui les résolvent.

Une implémentation de base du système nécessite les éléments suivants : des structures de données pour représenter les projets, les programmeurs et les rapport de bugs, une fonction pour créer un (rapport de) bug et l'assigner à un programmeur.

Ce système devra ensuite pouvoir marquer un bug comme doublon d'un autre bug.

Question 3.1 Structures de données [10 points]

Définissez une structure de données `Programmeur` permettant de représenter un programmeur. Un programmeur aura au moins :

- son nom ;
- son âge ;
- la liste des bugs qu'il gère.

Implémentez la liste des bugs sous forme d'une liste chaînée.

Définissez ensuite une structure de données `Projet` représentant un projet. Cette structure contiendra simplement le nom du projet.

Définissez finalement une structure de données `Bug` permettant de représenter un (rapport de) bug. Cette structure doit au moins contenir les éléments suivants :

- un entier représentant son identifiant (`id`) ;
- une description textuelle du bug ;
- un statut, qui peut prendre les valeurs suivantes : nouveau, abandonné, résolu et doublon ;
- le projet auquel ce bug appartient ;
- et potentiellement un autre bug dont il serait un doublon. Nous appellerons ce champ `duplicate`.

Plusieurs des éléments ci-dessus peuvent être représentés indirectement et nécessiter l'utilisation de pointeurs ou de structures auxiliaires. Vous pouvez par ailleurs ajouter à ces structures tout autre élément que vous jugez nécessaire. Expliquez alors de quoi il s'agit.

Note importante : concernant les différentes chaînes de caractères : on pourra ici considérer que les noms (programmeur et projet) sont de taille fixe, maximale (p.ex. 256) ; par contre, les descriptions de bugs devront être gérés *dynamiquement*, par copie, car ils sont plus variables en taille et potentiellement beaucoup plus gros.

Réponse Question 3.1 :

suite au dos ☞

Question 3.2 Création des bugs [10 points]

Ecrivez une fonction qui affecte une description à un bug.

Ecrivez ensuite une fonction qui crée un bug à partir d'un id, d'un projet et d'une description.

Réponse :

Question 3.3 Association de bugs à un programmeur [9 points]

Ecrivez une fonction qui ajoute un bug à un programmeur. Vérifiez que ce bug ne soit pas déjà assigné à ce programmeur.

Réponse :

Question 3.4 Doublons [3 points]

Créez une fonction qui définit un bug comme un doublon d'un autre bug. Cela modifie son statut et son champ `duplicate`. L'autre bug n'est par contre pas modifié.

Réponse :

suite au dos ☞

Question 3.5 Schéma mémoire [11 points]

Explicitez par un schéma la représentation en mémoire des objets correspondant à la situation suivante :

Un programmeur appelé Neo, âgé de 28 ans, travaille sur un projet nommé « WhiteRabbit ». Neo a 3 bugs à corriger pour ce projet :

- un premier bug, dont la description est « deja vu » ;
- un second bug, dont la description est « black cat », et qui est un doublon du premier bug ;
- un dernier bug, dont la description est « cookie ».

Note : Il n'est pas nécessaire de représenter ici les champs `id` et `statut` des bugs.

Réponse :