INFO0946 **Examen INFO0946** Août 2020	NOM : . . . . . PRÉNOM : . . . . . Matricule ULiège : **Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp**

Introduction à la Programmation (durée : 3h00)

Question	**Points**
Question 1	. . . . . /10
Question 2	. . . . . /20
Question 3	. . . . . /20
Consignes	

Consignes (à lire attentivement avant de répondre aux questions)

Suite à la crise sanitaire liée à l'épidémie de Covid-19, l'examen est organisé à distance via une session WebEx (pour poser des questions sur le questionnaire), via CAFÉ pour l'encodage de vos réponses (ainsi que la correction automatique) et la Plateforme de Soumission pour déposer vos réponses.

1. Le présent questionnaire et les différents fichiers annexes (e.g., les fichiers servant de canevas pour la soumission de vos réponses) sont disponibles sur eCampus (INFO0946, Sec. Examen).
2. Contrairement aux Challenges, CAFÉ ne fera pas de correction automatique avec feedback sur vos exercices. La seule chose que vérifiera CAFÉ, c'est l'encodage de vos réponses dans le canevas. Un email vous sera envoyé automatiquement avec le résultat de ces tests. Si jamais il y a des erreurs d'encodage, vous avez la possiblité de charger de nouveaux votre fichier (autant de fois que nécessaire, dans les limites du temps imparti par l'examen).
3. Il y a un fichier de canevas par question. Vous devez donc télécharger, depuis eCampus (Sec. Examen), trois fichiers textes (`question1.txt`, `question2.txt`, `question3.txt`). Ces fichiers doivent être complétés et placés dans une archive `zip` pour la soumission (voir page suivante).
4. La correction de l'examen sera réalisée automatiquement par CAFÉ, après l'examen en lui-même. Il est, dès lors, important que chacun de vos codes compile. Sinon, vous risquez une note nulle à la question ! !
5. La durée de l'examen est de 3h. N'attendez pas la dernière seconde pour soumettre votre fichier.

INFO0946 NOM : ..... PRÉNOM : .....

Soumission

Pour chacune des questions de cet examen, vous disposez, sur eCampus (Sec. Examen), d'un fichier texte (`question1.txt`, `question2.txt`, `question3.txt`) à compléter.

Vous devez soumettre ces trois fichiers textes, sous la forme d'une archive `zip`, sur la Plateforme de Soumission. Voici comment procéder sur les systèmes d'exploitation les plus courants. **Placez les trois fichiers à la racine de l'archive (donc pas dans un dossier).** [1]

Sous Windows Il suffit de cliquer sur le fichier à l'aide du bouton droit de la souris, sélectionner « Envoyer vers... » et sélectionner ensuite « Dossier compressé ».

Sous Linux (Ubuntu, Fedora, Linux Mint, ...) Il suffit de cliquer sur le fichier à l'aide du bouton droit de la souris, sélectionner « Compresser... ». Veillez bien à sélectionner « `.zip` » dans la liste des extensions possibles pour le fichier.

Sous OS X Cliquez sur le fichier en maintenant la touche Contrôle enfoncée (ou cliquez avec 2 doigts), sélectionnez « Compresser ».

Dans tous les cas Ne soumettez pas de fichier `.tar.gz`, `.7z`, `.rar` ou autre ! C'est bien un fichier `.zip` qui est attendu. Le nom de l'archive importe peu, tant que c'est une archive `zip` valide, dont le nom se termine bien par « `.zip` » **et ne comporte pas de caractères spéciaux comme des espaces, des parenthèses, etc.**

1. En Juin 2020, plus de la moitié des soumissions n'ont pas respecté cette règle. Simon & moi avons quand même pris le pli de modifier les soumissions (moyennant une « amende »). Ca ne sera plus forcément le cas lors de cette session !

Question 1 : Pointeurs ($10pts$)

`index` :	4000	2008
	⋮	⋮
	3008	76
	3004	1016
`z` :	3000	42
	⋮	⋮
	2016	789
	2012	1991
	2008	3004
	2004	5620
`data` :	2000	255
	⋮	⋮
	1020	7
	1016	1020
	1012	??
	1008	??
	1004	569
`value` :	1000	499

La première colonne donne le nom des variables, la deuxième donne des adresses mémoires (nous les exprimerons dans cette question en **base 10**) et, enfin, la troisième donne la valeur stockée à cette adresse (`??` signifie que la valeur est indéterminée). De plus, voici comment ont été déclarées les variables (on suppose que les entiers – `int` – sont représentés sur 4 octets [2])

```
1 int data, *index, value, z, *p = &z;
```

Dans cette question, vous devez indiquer la valeur des **expressions** suivantes. Considérez qu'entre chaque expression, la mémoire est réinitialisée telle que présentée dans le schéma. Si un accès mémoire est demandé à une adresse hors de l'intervalle $[1000, 4000]$, mentionnez l'erreur de segmentation par la valeur `SF` [3].

1. `value`
2. `*index`
3. `**index`
4. `***index`
5. `****index`
6. `*****index`
7. `&index`
8. `p - index`
9. `index - &data`
10. `&*p`
11. `*&data`
12. `p[0]`

2. les `short` sur 2, les `long sur 8`
3. Pour `S`egmentation `F`ault.

13. `p[-value]`
14. `++*p`
15. `*p++`
16. `*++p`
17. `value + 10`
18. `&data + 10`
19. `data &10`
20. `(short *) &data + 10`

Lors de votre soumission, vous devrez indiquer les valeurs de ces expressions, en fonction de l'état de la mémoire décrit ci-dessus. La façon de formuler vos réponses est indiquée dans le fichier servant de canevas à votre soumission, fichier disponible sur eCampus (Sec. Examen – `question1.txt`).

Question 2 : Construction par Invariant ($20pts$)

Soit `tab`, un tableau d'entiers de taille `N+1`, où `N` est une constante entière déclarée et initialisée dans le code. On considère le tableau `tab` comme étant **déjà rempli sur les `N` premières cases** (la valeur de la dernière case étant indéterminée). Ainsi, si `N` vaut 5, le tableau `tab` peut ressembler à ceci :

	0				5
`tab` :	4	3	6	2	5	??

Dans cette question, on demande d'écrire le code permettant d'insérer une valeur entière `x`, donnée, à la position `indice` (par hypothèse, on considère que `indice` appartient à l'intervalle $[0, N]$), elle aussi donnée. Les valeurs suivant la position `indice` devront être décalée d'une case vers la droite pour permettre l'insertion. Ainsi, par exemple, pour le tableau `tab` défini ci-dessus, une valeur `x=-1` et `indice=3`, on obtient :

	0					5
`tab` :	4	3	6	-1	2	5

Attention, pour résoudre ce problème, vous devez respecter ces contraintes :
- vous ne pouvez utiliser qu'une et une seule boucle de type `while` ;
- vous ne pouvez pas utiliser d'instruction conditionnelle (dans et en dehors de la boucle) ;
- vous devez viser une complexité théorique $\mathcal{O}(N)$. De plus, vous ne pouvez entrer dans votre boucle, qu'au plus N fois. Attention, le nombre d'itérations sera compté !
- Rien ni personne ne vous demande d'écrire quoi que ce soit à l'écran : **ne polluez donc pas la sortie standard sous peine d'empêcher la correction de votre travail !**

Le squelette de votre code est le suivant :

```c
#include <stdio.h>

int main(){
  const unsigned int N = ...;
  int tab[N+1];
  int x = ...;
  int indice = ...;

  /*
   * Code de remplissage du tableau tab. Ce code ne vous est pas donne.
   */

  // Votre code viendra ici (variables + instructions)

}//fin programme
```

Vous devez considérer, lors de la soumission, que votre extrait de code est copié et collé[4] à l'endroit indiqué par le commentaire « `Votre code viendra ici (variables + instructions)` ». Dès lors, redéclarer la constante `N`, le tableau `tab` ou encore les variables `x` ou `indice` fera échouer la compilation.

Lors de votre soumission, vous devrez fournir

1. l'Invariant de votre boucle ;
2. la Fonction de Terminaison de votre boucle ;
3. le code C.

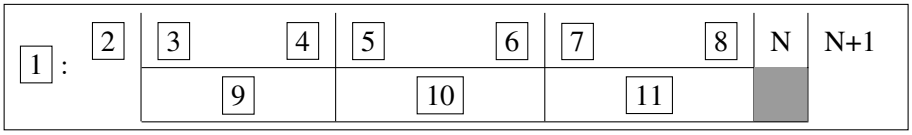
La façon de formuler votre Invariant et votre Fonction de Terminaison est indiqué dans les sections ci-dessous. Le fichier servant de canevas à votre soumission est disponible sur eCampus (Sec. Examen, – `question2.txt`). En outre, le fichier `question2.c` est disponible sur eCampus (Sec. Examen). Il vous permet de tester la compilation de votre code (sans devoir tout réécrire par vous-mêmes). Attention, les valeurs données pour `N`, `tab`, `x` et `indice` données dans `question2.c` sont des cas particuliers.

4. En fait, c'est à quelques détails près le cas.

Question 2.A : Invariant

Ce problème peut être résolu sans découpe en sous-problème.

Voici un Invariant muet[5] :



Selon l'équation fondamentale des Invariants Graphiques des codes qui manipulent des tableaux, i.e., :

$$1 \text{ Tableau} = 1 \text{ Dessin},$$

cet Invariant doit représenter le tableau manipulé par votre code. Les boites **2.** à **8.** servent à mentionner des indices de ces tableaux. Ces boites doivent donc être remplacées par des constantes ou des noms de variables.

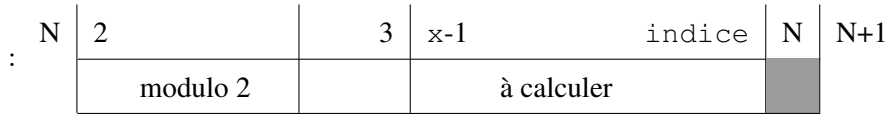
Toutes les boites ne doivent pas forcément être précisées : si vous indiquez que la valeur de la boite 4. est « k », il est implicite que la valeur de la boite 5. est « $k + 1$ » : inutile alors de le préciser.

Pour les autres boites, les choix disponibles sont résumés dans la table 1.

Pour les boites 9, 10 et 11
1. déjà parcouru ;
2. à calculer ;
3. j'ai déjà inséré x ;
4. décalé d'une case vers la gauche ;
5. décalé d'une case vers la droite ;
6. $\cdot = tab - 1$
7. modulo 2.
_ (Voir la remarque plus bas)

TABLE 1 – Invariant : Possibilités de choix pour les boites **9** à **11**.

Par exemple, si vous pensez qu'un bon Invariant serait :



Alors, complétez le fichier de réponse comme suit :

```
1. _
2. N
3. 2
4. _
5. _
6. 3
7. x-1
8. indice
9. 6
```

5. Tout ce qui est dans le cadre fait partie de l'Invariant

```
10. _
11. 2
```

Vous voyez qu'il est possible d'affecter une variable ou une constante d'un modificateur +1 ou -1. Il suffit dans ce cas d'écrire ce +1 ou -1 comme montré ci-dessus. N'introduisez pas de parenthèses.

Dans tous les cas, si vous pensez que la bonne réponse consiste à ne rien écrire à l'emplacement d'une boite, n'inscrivez rien comme réponse ou un « _ »[6].

Pour votre facilité, (et pour les distraits qui en oublieraient un), les numéros des 11 boites sont déjà inscrits dans le fichier de réponse.

Question 2.B : Fonction de Terminaison

Dand cette question, nous vous demandons de fournir la Fonction de Terminaison de votre boucle.

Dans le fichier de réponses, à la question sur la Fonction de Terminaison, nous vous demandons de nous la fournir sous la forme d'une **expression C** valide.

La correction s'assurera entre autres :
— Que cette expression utilise des variables de votre code ;
— Que son domaine est bien l'ensemble des Entiers (positifs si le gardien est vrai) ;
— Que sa valeur décroit strictement entre deux itérations.

Si vous pensez que la Fonction de Terminaison de votre code, qui manipule la variable `toto` et la constante `G` devrait être :

$$t = \texttt{G} + 17 - \texttt{toto}$$

Encodez :

```
G + 17 - toto
```

N'indiquez pas « `F =` » ou « `t =` » mais seulement l'expression qui sert à calculer la valeur de votre Fonction de Terminaison. Il est inutile de rappeler que votre expression entière est positive en rajoutant un « > 0 » (en plus, cela transforme votre expression en prédicat et cela rend votre réponse incorrecte).

6. Caractère « underscore ».

Question 3 : Programme Complet ($20pts$)

Dans cet exercice, nous nous intéressons au transport maritime de marchandises. L'idée est que diverses marchandises (décrites par leur volume, poids et nom) sont stockées dans des containers, eux-mêmes placés dans un navire.

Un squelette de programme, basé sur les principes de la compilation séparée, a été créé pour vous. Il est composé de deux fichiers : le header (`question3.h`) et le module (`question3.c`). Ces deux fichiers sont disponibles sur eCampus (Sec. Examen). Votre objectif est de compléter le code existant et de soumettre vos réponses dans le fichier texte correspondant (`question3.txt`, disponible sur eCampus, Sec. Examen).

1. `question3.h`. C'est le fichier d'en-tête contenant les déclarations des structures de données et les prototypes des différentes fonctions et procédures. L'extrait de code 1 vous donne son contenu.
2. `question3.c`. Il s'agit de l'implémentation du header `question3.h`. Les fonctions `cree_navire()` et `cree_marchandise()` ont déjà été implémentées pour vous. Vous devez donc les utiliser quand vous le jugerez nécessaire. L'extrait de code 2 vous donne le squelette du fichier `question3.c`. En outre, le fichier `question3.c` contient une fonction `main()` reprenant le programme principal. Cette fonction `main()` est importante pour la Question 3.C.

Extrait de code 1 – Fichier `question3.h`

```
#ifndef __QUESTION3__
#define __QUESTION3__

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

//Charge maximum du navire
#define MAX_CHARGE_NAVIRE 1250000
#define MAX_CARACTERES 100

//Définition d'une marchandise
typedef struct{
  float volume; //exprimé en m^3
  float poids;  //exprimé en tonne
  char nom[MAX_CARACTERES+1];
  unsigned int id; //\in [0, 2^32 - 1]
}Marchandise;

//Définition d'un container
typedef struct{
  unsigned int nb_marchandises;
  Marchandise *tab_marchandises;
  double volume_total;
}Container;

//Définition d'un navire
typedef struct{
  unsigned int nb_containers;
  Container *tab_containers;
}Navire;

/*
 * Cree une marchandise sur base d'informations lues au
 * clavier.
 *
 * @pré: /
 * @post: un pointeur vers une marchandise valide.  NULL en cas d'erreur.
 */
Marchandise *cree_marchandise();

/*
```

```
 * Question 3.A
 *
 * Crée un container.
 */
Container *cree_container(unsigned int nb_marchandises);

/*
 * Question 3.B
 *
 * Ajoute, si possible, une marchandise au container.
 */
int ajout_marchandise_container(Container *c, Marchandise *m);

/*
 * Question 3.C
 *
 * Determine la charge d'un container.
 */
float charge_container(Container *c);

/*
 * Question 3.D.
 *
 * Sauve, dans un fichier, la liste des des marchandises transportées par le navir.
 *
 * Format:
 * # containers
 * Container 1:
 *      # marchandises
 *      id; nom; volume; poids
 *      id; nom; volume; poids
 *      ...
 *      nom; volume;poids
 * Container 2:
 *      # marchandises
 *      id; nom; volume; poids
 *      id; nom; volume; poids
 *      ...
 *      id; nom; volume;poids
 * ...
 * Container n:
 *      # marchandises
 *      id; nom; volume; poids
 *      id; nom; volume; poids
 *      ...
 *      id; nom; volume;poids
 * Exemple:
 * 2
 * Container 1:
 *      3
 *      34567; guitares; 10.4; 0.009
 *      1234; amplificateurs; 105.23; 0.325
 *      98705; batteries; 231.03; 0.657
 * Container 2:
 *      2
 *      782180; sono; 1245.032; 0.345
 *      234709; light; 567.03; 0.342
 *
 * @pre: n!=NULL
 * @post: 1 si le navire a pu etre sauve dans le fichier.
 *        -1 sinon
 */
int sauve_navire(Navire *n, char *fichier);

```

```
/*
 * Question 3.E.
 *
 * Détermine, pour un navire, la charge moyenne par container et le nombre
 * total de marchandises transportées.
 */

#endif
```

Extrait de code 2 – Fichier `question3.c`

```
#include "question3.h"

/*
 * Variables globales permettant l'implémentation rapide de certaines
 * fonctionnalités non fournies
 */
Marchandise test_marchandise = {2.4, 0.56, "Guitares", 12345};
Marchandise m[3] = {{2.4, 0.56, "Guitares", 12345}, {3.7, 0.98, "Amplis", 4567},
                    {4.2, 1.2, "Batterie",  98076}};
Container test_container = {3, m, 10.3};
Navire test_navire = {1, &test_container};

Marchandise *cree_marchandise(){
    return &test_marchandise;
}//fin cree_marchandise()


Container *cree_container(unsigned int nb_marchandises){
  //VOTRE CODE ICI
}//fin cree_container()

int ajout_marchandise_container(Container *c, Marchandise *m){
  //VOTRE CODE ICI
}//fin ajout_marchandise_container()

float charge_container(Container *c){
  //VOTRE CODE ICI
}//fin charge_container()

int sauve_navire(Navire *n, char *fichier){
  //VOTRE CODE ICI
}//fin sauve_navire()

//QUESTION 3.C

static Navire *cree_navire(unsigned int nb_containers){
  return &test_navire;
}//fin cree_navire()

int main(){
  Navire *n;
  Marchandise *m;
  unsigned int i;
  float charge_moyenne;
  unsigned int total_marchandise;

  //Cree un navire pouvant contenir 1 containers
  n = cree_navire(1);
  if(n==NULL)
    return -1;

  /*
   * STATISTIQUES sur le navire (utilisation de la fonction/procédure de
```

```
    * la Question 3.E)
    * On veut afficher à l'écran la charge moyenne d'un container et le nombre
    * total de marchandises transportées par le navire.
    * Les variables nécessaires ont déjà été déclarées dans le main().  Vous ne
    * pouvez utiliser rien d'autre.
    */
  //L'invocation de votre fonction vient ici
  printf("La charge moyenne d'un container est: %f\n", charge_moyenne);
  printf("La quantité totale de marchandise est: %u\n", total_marchandise);

  return 1;
}//fin programme
```

Dans toutes les sous-questions et quand cela s'avère nécessaire, il est demandé d'appliquer les principes de la programmation défensive.

En outre, vous pouvez toujours invoquer n'importe quelle fonctionnalité de la gestion des navires (cf. `question3.h`), même si vous ne l'avez pas implémentée.

Question 3.A

Implémentez la fonction `Container *cree_container(unsigned int nb_marchandises);`. Cette fonction crée un container du navire pouvant accueillir au maximum `nb_marchandises` (> 0). Cette fonction retourne un pointeur vers un container vide ou `NULL` en cas de problème.

Insérez le corps de votre fonction dans le fichier `question3.txt`. Il est inutile d'ajouter des directives de prépocessing (`#include`, `#define`, etc.), de réécrire la signature de la fonction ainsi que les accolades représentant le bloc du corps de la fonction. Vous ne pouvez pas non plus redéclarer les paramètres formels de la fonction. Faites attention à ne pas utiliser l'underscore (_) dans vos noms de variables (surtout en début de nom : c'est interdit par le standard du langage C).

Question 3.B

Implémentez la fonction `int ajout_marchandise_navire(Navire *n, Marchandise *m);` Cette fonction a pour but de trouver, dans le navire `n`, un container pouvant stocker la marchandise `m`. Un container pourra stocker la marchandise uniquement si le volume encore disponible dans le container est supérieur au volume de la marchandise. Si un container est trouvé, la marchandise est ajoutée dans ce container et un nombre entier positif (i.e., 1) est retourné. Dans le cas contraire, une valeur entière négative (i.e., -1) est retournée. Pensez à utiliser les fonctionalités mises à votre disposition.

Insérez le corps de votre fonction dans le fichier `question3.txt`. Il est inutile d'ajouter des directives de prépocessing (`#include`, `#define`, etc.), de réécrire la signature de la fonction ainsi que les accolades représentant le bloc du corps de la fonction. Vous ne pouvez pas non plus redéclarer les paramètres formels de la fonction. Faites attention à ne pas utiliser l'underscore (_) dans vos noms de variables (surtout en début de nom : c'est interdit par le standard du langage C).

Question 3.C

Implémentez la fonction `float charge_container(Container *c);`. Cette fonction retourne la charge totale des marchandises contenues dans un container. La charge d'un container correspond au poids total des marchandises dans le container.

Insérez le corps de votre fonction dans le fichier `question3.txt`. Il est inutile d'ajouter des directives de prépocessing (`#include`, `#define`, etc.), de réécrire la signature de la fonction ainsi que les accolades représentant le bloc du corps de la fonction. Vous ne pouvez pas non plus redéclarer les paramètres formels de la fonction. Faites attention à ne pas utiliser l'underscore (_) dans vos noms de variables (surtout en début de nom : c'est interdit par le standard du langage C).

Question 3.D

Implémentez la fonction `int sauve_navire(Navire *n, char *fichier);`. Cette fonction sauve, dans un fichier donné, tous les containers (et leurs marchandises) transportés par le navire. Le format du fichier est le suivant :

```
nb_containers
Container 1:
   nb_marchandises
   id; nom; volume; poids
   id; nom; volume; poids
   ...
   id; nom; volume; poids
Container 2:
   nb_marchandises
   id; nom; volume; poids
   id; nom; volume; poids
   ...
   id; nom; volume; poids
...
Container n:
   nb_marchandises
   id; nom; volume; poids
   id; nom; volume; poids
   ...
   id; nom; volume; poids
```

Par exemple, pour un navire transportant 2 containers :

```
2
Container 1:
   3
   34567; guitares; 10.4; 0.009
   1234; amplificateurs; 105.23; 0.325
   98705; batteries; 231.03; 0.657
Container 2:
   2
   782180; sono; 1245.032; 0.345
   234709; light; 567.03; 0.342
```

La fonction retourne l'entier 1 si les informations ont pu être correctement sauvées dans le fichier, -1 dans le cas contraire.

Insérez le corps de votre fonction dans le fichier `question3.txt`. Il est inutile d'ajouter des directives de préprocessing (`#include`, `#define`, etc.), de réécrire la signature de la fonction ainsi que les accolades représentant le bloc du corps de la fonction. Vous ne pouvez pas non plus redéclarer les paramètres formels de la fonction. Faites attention à ne pas utiliser l'underscore (_) dans vos noms de variables (surtout en début de nom : c'est interdit par le standard du langage C).

Question 3.E

Écrivez une fonction ou procédure dont le nom est `comptage` et qui retourne plusieurs statistiques sur le navire. Les statistiques qui nous intéressent sont la charge moyenne d'un container et le nombre total de marchandises transportées par le navire.

Insérez le prototype et le corps de votre fonction ou procédure dans le fichier `question3.txt`. Il est inutile d'ajouter des directives de préprocessing (`#include`, `#define`, etc.). Faites attention à ne pas utiliser l'underscore (_) dans vos noms de variables (surtout en début de nom : c'est interdit par le standard du langage C).

Question 3.F

Écrivez la ligne correspondant à l'appel de votre fonction ou procédure `comptage`, c'est-à-dire l'instruction qui doit remplacer le commentaire « `L'invocation de votre fonction vient ici` » à la ligne 53 de l'extrait de code 2.

Insérez l'instruction dans le fichier `question3.txt`. N'écrivez que l'instruction correspondant à l'invocation de votre fonction ou procédure `comptage`.