INFO0946 **Examen INFO0946** Janvier 2021	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	. . . . . /15
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. Une session Collaborate est mise en place pour poser des questions sur le questionnaire. Vous devez vous connecter obligatoirement à cette session, que vous ayez des questions ou pas. Elle vous permettra de bénéficier des réponses aux questions des autres étudiants (ou des divers conseils/rappels durant l'examen). CAFÉ est utilisé pour l'encodage de vos réponses (ainsi que la correction automatique) et la Plateforme de Soumission pour déposer vos réponses.

Si vous souhaitez signer l'examen, il vous suffit d'uploader, sur la Plateforme de Soumission, le fichier de canevas des réponses (`info0946-examen.c` – voir page suivante pour les détails de soumission) sans y ajouter vos réponses.

1. Le présent questionnaire et les différents fichiers annexes (e.g., le fichier servant de canevas pour la soumission de vos réponses) sont disponibles sur eCampus (INFO0946, Sec. Examen Janvier).
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 unique pour les trois questions. Vous devez le télécharger depuis eCampus (INFO0946, Sec. Examen Janvier). Ce fichier doit être complété et placé 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 (mais ne passez pas toute la durée de l'examen à essayer de compiler le code. . .) ! !
5. La durée de l'examen est de 3h00. N'attendez pas la dernière seconde pour soumettre votre fichier (**il n'y aura aucune extension de deadline**).

Soumission

Pour répondre aux questions de cet examen, vous disposez, sur eCampus (INFO0946, Sec. Examen Janvier), d'un fichier (`info0946-examen.c`) à compléter.

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

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.**

Question 1 : Pointeurs ($10pts$)

	340	27
`data:`	336	208
		...
		...
	220	3
	216	17
	212	5
	208	220
`x:`	204	336

Le dessin sur la gauche donne l'état de la mémoire. La première colonne indique le nom des variables, la deuxième des adresses mémoires et, enfin, la troisième donne la valeur stockée à cette adresse. De plus, voici comment ont été déclarées les variables (on suppose que les entiers sont représentés sur 4 octets) :

```c
int data, x, *ptr = &x;
char *E = "Session", *F = E + 7;
```

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 $[204, 340]$, mentionnez l'erreur de segmentation par la valeur `SF` [1].

1. `data`
2. `*data`
3. `**data`
4. `&data`
5. `&*data`
6. `*&x`
7. `*ptr++`
8. `+++&data`
9. `*++ptr`
10. `ptr + data`
11. `&data - ptr`
12. `(short *) ptr + x`
13. `x & *ptr`
14. `*F`
15. `F - E`
16. `**F`
17. `*(x = data)`
18. `*(x == data)`
19. `F[-4] - *(E + 2)`
20. `E[**data + 2] - F['E' - 'F']`

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 (INFO0946, Sec. Examen Janvier– `info0946-examen.c`).

1. Pour `S`egmentation `F`ault.

Question 2 : Construction par Invariant ($15pts$)

Soit la fonction suivante :

Extrait de code 1 – Interface de la fonction `test_p()`

```
/*
 * PréCondition : /
 * PostCondition : test_p vaut 1 si p(x) est vrai.  0 sinon.
 */
int test_p(int x);
```

Elle permet de tester que le paramètre `x` vérifie une certaine propriété p. Par exemple, si la propriété p correspond à "est divisible par 5", alors le code suivant :

```
int x = 10;
if(test_p(x))
  printf("Vrai!\n");
else
  printf("Faux!\n");
```

affichera la chaîne de caractère "Vrai" à l'écran.

Dans cette question, on dispose d'un tableau `t` à `N` valeurs entières ($N > 0$). Par exemple, si `N=8` :

	0						7
t :	16	-1	7	4	-12	9	2	13

FIGURE 1 – Exemple de tableau `t` initial.

Dans cette question, on souhaite implémenter la fonction suivante :

Extrait de code 2 – Interface de la fonction `filtrer`

```
/*
 * PréCondition : t valide, N>0
 * PostCondition : filtrer retourne le nombre de valeurs dans t tel que p(·) et t filtré
 *                 en fonction de p(·).
 */
int filtrer(int *t, unsigned int N);
```

La fonction `filtrer()` filtre le tableau `t` en fonction d'une propriété p, vérifiée par la fonction `test_p()` (voir Extrait de Code 1). Le filtre consiste à modifier `t` de sorte que les valeurs de `t` qui vérifie la propriété p se trouve en fin de tableau (i.e., dans la partie droite de `t` – peu importe où exactement, l'ordre n'a pas d'importance), tandis que les valeurs qui ne vérifient pas p se retrouve en début de tableau (i.e., dans la partie gauche de `t` – peu importe où, l'ordre n'a pas d'importance). En outre, `filtrer()` retourne le nombre de valeurs dans `t` qui vérifient la propriété p.

Exemple 1 : si la propriété p correspond à "est négatif" et qu'on applique `filtrer()` sur le tableau `t` (voir Fig. 1), on obtient le tableau `t` modifié suivant :

	0						7
t :	16	13	7	4	2	9	-12	-1

FIGURE 2 – Tableau `t` après l'invocation `filtrer(t, 8);` pour p 'est négatif". La fonction retourne la valeur 2.

et `filtrer()` retourne la valeur 2 (seules 2 valeurs de `t` vérifient la propriété p).

Exemple 2 : si la propriété p correspond à "est pair" et qu'on applique `filtrer()` sur le tableau `t` (voir Fig. 1), on obtient le tableau `t` modifié suivant :

	0							7
t :	13	-1	7	9	-12	2	4	16

FIGURE 3 – Tableau `t` après l'invocation `filtrer(t, 8);` pour p 'est pair". La fonction retourne la valeur 4.

et `filtrer()` retourne la valeur 4 (seules 4 valeurs de `t` vérifient la propriété p).

Exemple 3 : si la propriété p correspond à "est plus grand ou égal à 20" et qu'on applique `filtrer()` sur le tableau `t` (voir Fig. 1), on obtient le tableau `t` modifié suivant :

	0							7
t :	16	-1	7	4	-12	9	2	13

FIGURE 4 – Tableau `t` après l'invocation `filtrer(t, 8);` pour p 'est plus grand ou égal à 20". La fonction retourne la valeur 0.

le tableau `t` n'est pas modifié et `filtrer()` retourne la valeur 0 (aucune valeur de `t` ne vérifie la propriété p).

Nous vous demandons d'implémenter la fonction `filtrer()` en respectant les contraintes suivantes :

— votre solution doit être construite autour d'**une et une seule boucle de type `while`**;
— votre code doit avoir une complexité en $O(N)$;
— il est interdit de passer par un tableau intermédiaire;
— il n'est pas nécessaire de pratiquer la programmation défensive;
— vous ne pouvez utiliser aucune librairie;
— vous ne devez pas implémenter la fonction `test_p()` mais vous devez forcément l'invoquer dans le code de `filtrer()`.

Lors de votre soumission, vous devrez fournir

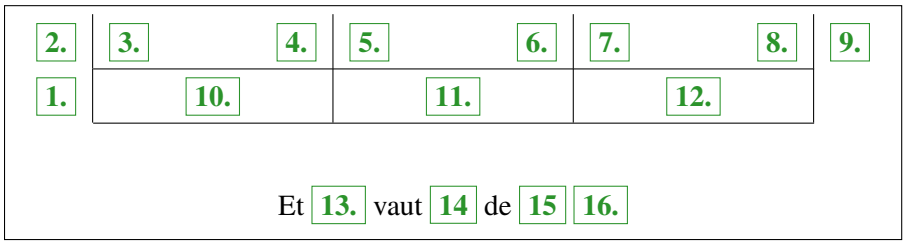
1. l'Invariant Graphique de votre boucle (Question 2.A);
2. la Fonction de Terminaison de votre boucle (Question 2.B);
3. le code C (Question 2.C).

La façon de formuler votre Invariant Graphique et votre Fonction de Terminaison est indiqué dans les sections ci-dessous. Le fichier servant de canevas à votre soumission est disponible sur eCampus (INFO0946, Sec. Examen Janvier, – `info0946-examen.c`). 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` et `t`, ainsi que l'implémentation de `test_p()` donnés dans `question2.c` sont des cas particuliers.

Attention, soyez prudent sur la gestion de votre temps. Ne perdez pas de temps à "hacker" le fichier `question2.c` pour obtenir une version (forcément) bancale de votre solution. Si vous appliquez correctement les principes de la programmation par Invariant de Boucle, tout devrait se bien dérouler.

Question 2.A : Invariant Graphique

Voici un Invariant Graphique muet[2] :



Les boites **1.** à **9.** et **13.** servent à mentionner des indices. 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 est « $k + 1$ » : inutile alors de le préciser.

Pour les autres boites (**10.** à **12.** et **14.** à **16.**), les choix disponibles sont résumés dans le tableau 1.

Boites 10. à 12.	**Boite 14.**	**Boite 15.**	**Boite 16.**
1. À Vérifier	1. la division	1. valeurs	1. vérifiées
2. Déjà Vérifié	2. la multiplication	2. itérations	2. parcourues
3. À parcourir	3. le reste	3. décisions	3. Nakamura[3]
4. Déjà parcouru	4. le nombre	4. boucles	4. affichées
5. À Afficher	5. la chanson	5. Aya	5. filtrées
6. Déjà Affiché	6. la somme	6. fonctions	6. remplies
7. À Filtrer	7. le nombre premier	7. données	7. prises
8. Déjà Filtré	8. le filtre	8. variables	8. multipliées
9. À Remplir	9. la vie	9. directions	9. données
10. Déjà rempli	10. la Force	10. impressions	10. testées

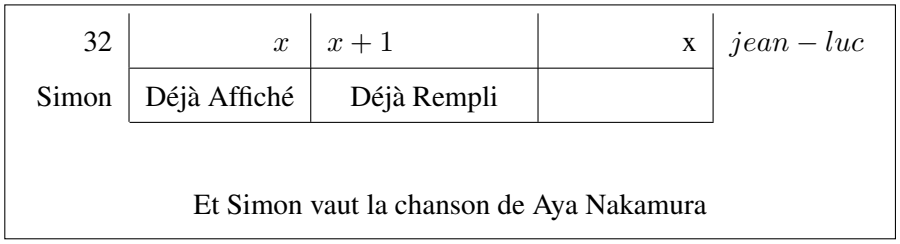
TABLE 1 – Invariant Graphique : Possibilités de choix pour les boites **10.** à **12.** et **14.** à **16.**

Attention, nous inspecterons votre code pour détecter s'il respecte l'Invariant Graphique ainsi formé.

Encodage dans le Fichier de Soumission

Veuillez indiquer, dans le fichier de soumission (`info0946-examen.c`), à la réponse sur l'Invariant Graphique, le numéro de la boite, suivi d'un point, suivi de la valeur numérique ou du nom de variable ou constante de votre choix (pour les boites de **1.** à **9.** et **13.** ou d'un nombre entre 1 et 10 correspondant à votre choix (pour les boites **10.** à **12.** et **14.** à **16.**).

Si vous pensez qu'un bon Invariant Graphique serait :



Alors[4], complétez le fichier de réponse comme suit :

2. Tout ce qui est dans le cadre fait partie de l'Invariant Graphique
3. Selon Wikipedia (mais seulement selon Wikipedia. . .), « Aya Nakamura [...] est une autrice-compositrice-interprète française. »
4. Vous êtes conscient·e que ce n'est pas la bonne réponse ?

```
1. Simon
2. 32
3. _
4. x
5. x+1
6. _
7. _
8. x
9. jean - luc
10. 6
11. 10
12. _
13. Simon
14. 5
15. 5
16. 3
```

Il est toujours 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. De même, il est possible d'additionner ou de soustraire deux identifiants ou constante numériques.

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 « _ ».

Pour votre facilité, (et pour les distraits qui en oublieraient un), les numéros des 16 boites sont déjà inscrits dans le fichier de réponse.

Fonction de Terminaison

La Fonction de Terminaison[5] permet de fournir la preuve que la boucle se termine. Dans cette question, nous vous demandons de fournir la Fonction de Terminaison de votre boucle.

Dans le fichier de réponses, aux questions 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.

Exemple

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
```

5. À ne confondre ni avec le Critère d'Arrêt, ni avec le Gardien de Boucle

N'indiquez pas « `F =` » ou « `t =` » mais seulement l'expression qui sert à calculer la valeur de votre Fonction de Terminaison. De plus, n'ajouter pas « > 0 » à votre soumission. en effet, $G + 17 - toto > 0$ n'est pas une fonction à valeur entière mais Booléenne [6].

Question 2.C : Code

Donnez le *code* complet de `filtrer()`. Il n'est pas demandé de pratiquer la programmation défensive.

Dans le fichier `info0946-examen.c`, il suffit d'inclure le corps de `filtrer()`, c'est à dire ce qui remplace la ligne « `// Votre code ici` » dans l'extrait de code 3. Vous ne devez pas remettre le prototype ni les accolades ouvrante et fermante des lignes 1 et 3.

Extrait de code 3 – Squelette de `filtrer` à compléter

```
1 int filtrer(int *t, unsigned int N){
2    // Votre  code ici
3 }
```

6. Et donc c'est incorrect...

Question 3 : Programmation Modulaire ($25pts$)

Transformer sa maison en chambre d'hôtes, beaucoup en rêvent, vous l'avez fait.[7] Après plusieurs années passées dans l'industrie, vous décidez de changer de vie. A cette fin, vous venez d'acquérir plusieurs bâtiments dans le Sud de la France. Ces bâtiments, après rénovation, vous permettent de vous lancer dans une activité de chambres d'hôtes. Afin de gérer votre clientèle, vos diverses chambres et toutes les réservations, vous décidez d'implémenter un programme de gestion de vos chambres.

Un squelette de programme, basé sur les principes de la compilation séparée, a été créé pour vous (la Fig. 5 vous donne une vision de haut niveau de l'architecture de votre code ainsi que les relations entre les différents fichiers sources). Votre objectif est donc de compléter le code existant.

Le programme est constitué de trois fichiers sources :

1. `chambre.h`. C'est le fichier header contenant les déclarations des structures de données et les prototypes des différentes fonctions et procédures. L'extrait de code 4 vous donne son contenu.
   — la macro `MAX_CARACTERES` indique le nombre maximum de caractères utiles dans une chaîne de caractères manipulée par votre programme.
   — On définit vos chambres d'hôtes via la structure `ChambresHotes`. La structure est composées d'un tableau de clients (champ `tab_clients`) et la taille de ce tableau (champ `nb_clients`, > 0), chaque élément représentant un client de vos chambres. En outre, la structure est aussi composée d'un certain nombre de chambres qu'un `Client` peut louer représentées par un tableau de `Chambre` (champ `tab_chambres`) et sa taille (champ řnb_chambres, > 0).
2. `chambre.c`. Il s'agit de l'implémentation du header `chambre.h`. Les fonctions `charge_clients()`, `charge_chambres()` et `date_courante()` ont déjà été implémentées. Vous devez donc les utiliser quand vous le jugerez nécessaire. L'extrait de code 5 vous donne le squelette du fichier `chambre.c`.
3. `main-chambre.c`. C'est le programme principal, il sera l'objet de la Question 3.D.

FIGURE 5 – Architecture du code.

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 des chambres d'hôtes (cfr. `chambre.h`), même si vous ne l'avez pas implémentée.

Extrait de code 4 – Fichier `chambre.h`

```
#define MAX_CARACTERES 100

typedef struct{
  unsigned short annee;
  unsigned short mois;
  unsigned short jour;
}Date;

typedef struct{
  unsigned short id_chambre; //identifiant de la chambre [0; 2^16 - 1]
  char nom[MAX_CARACTERES+1]; //nom de la chambre
  unsigned short nb_lits; //nombre de lits dans la chambre
  unsigned short nb_personnes; //nombre de personnes dans la chambre
  Date date_debut; //début de la location
```

7. "Bienvenue chez Nous", ©TF1.

```
  Date date_fin; //fin de la location
  struct client_t *locataire; //locataire de la chambre (NULL si non louée)
  float prix; //prix par jour;
}Chambre;

typedef struct client_t{
  unsigned long id_client; //identifiant du client [0; 2^32 - 1]
  char prenom[MAX_CARACTERES+1];
  char nom[MAX_CARACTERES+1];
  char email[MAX_CARACTERES+1];
  Chambre *chambre_louee; //chambre louée (NULL si pas de location en cours)
}Client;

typedef struct{
  Chambre *tab_chambres;
  unsigned int nb_chambres;
  Client *tab_clients;
  unsigned int nb_clients;
}ChambresHotes;

/*
 * Calcule le nombre de jours séparant 2 dates.
 * PréCondition : d1 valide, d2 valide, d1 >= d2
 * PostCondition : difference_date retourne le nombre de jours séparant les deux dates.
 */
int difference_date(Date *d1, Date *d2);

/*
 * Charge, depuis un fichier, les clients des chambres d'hôtes. Les clients sont
 * placés dans l'objet ch.
 * PréCondition : fichier_clients valide, ch valide, ch->tab_clients valide.
 * PostCondition : charge_client retourne 1 si les clients ont pu être enregistrés dans ch.
 *                 -1 en cas d'erreur
 */
int charge_clients(char *fichier_clients, ChambresHotes *ch);

/*
 * Charge, depuis un fichier, les chambres de la chambre d'hôtes. Les chambres
 * sont placées dans l'objet ch.
 * PréCondition : ficher_chambres valide, ch valide, ch->tab_chambres valide.
 * PostCondition : charge_chambres retourne 1 si les chambres ont pu être enregistrées dans ch.
 *                 -1 en cas d'erreur
 */
int charge_chambres(char *fichier_chambres, ChambresHotes *ch);

/*
 * Question 3.A
 */
ChambresHotes *cree_chambres_hote(char *fichier_clients, char *fichier_chambres,
                                  unsigned int nb_chambres, unsigned int nb_clients);

/*
 * Question 3.B
 */
int sauve_clients(ChambresHotes *ch, char *fichier);

/*
 * Question 3.C (calcul de statistiques sur les chambres d'hôtes)
 */
```

Extrait de code 5 – Fichier `chambre.c`

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

```c
#include <assert.h>

#include "chambre.h"

int difference_date(Date *d1, Date *d2){
  //CODE NON DONNÉ
}//fin difference_date()

int charge_clients(char *fichier_clients, ChambresHotes *ch){
  //CODE NON DONNÉ
}//fin charge_clients()

int charge_chambres(char *fichier_chambres, ChambresHotes *ch){
  //CODE NON DONNÉ
}//fin charge_chambres

//à completer par vos soins
```

Les fichiers composants votre programme (`chambre.h`, `chambre.c` et `main-chambre.c`) vous sont fournis sur eCampus (INFO0946, Sec. Examen Janvier). Vous pouvez les utiliser pour tester la compilation de votre code. Les modules `difference_date()`, `charge_clients()` et `charge_chambres()` ont été implémentés pour vous à l'aide de variables globales. Ne les modifiez pas. Elles sont là pour vous permettre de compiler et tester votre code.

Attention, soyez prudent sur la gestion de votre temps. Ne perdez pas de temps à "hacker" les fichiers `chambre.c` et `main-chambre.c`. Réfléchissez bien avant de commencer à coder. Cela devrait vous faire gagner du temps par la suite.

Question 3.A

Implémentez la fonction `ChambresHotes *cree_chambres_hote(char *fichier_clients, char *fichier_chambres, unsigned int nb_chambres, unsigned int nb_clients);` dans le fichier `chambre.c`. Cette fonction crée votre sytème de gestion de chambres d'hôtes qui comporte `nb_chambres` (ne peut être nul) et `nb_clients` (ne peut être nul). Les détails relatifs à chaque chambre doivent être chargés depuis un fichier dont le nom est contenu dans `fichier_chambres`. De la même manière, les informations relatives aux clients doivent être chargées depuis le fichier dont le nom est contenu dans `fichier_clients`. La fonction renvoie `NULL` en cas de problème.

Insérez le corps de votre fonction dans le fichier `info0946-examen.c`. 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.B

Implémentez la fonction `int sauve_clients(ChambresHotes *ch, char *fichier)` dans le fichier `chambre.c`. Cette fonction sauve, dans un fichier donné, toutes les informations relatives aux clients de vos chambres d'hôtes. Le format du fichier est le suivant :

```
nb_clients
id_client; prenom nom; email; id_chambre (si en location)
id_client; prenom nom; email; id_chambre (si en location)
...
id_client; prenom nom; email; id_chambre (si en location)
```

Par exemple, pour trois clients dont deux (Laurent Mathy et Benoit Donnet) louent actuellement une chambre :

```
3
1234567; Simon Liénardy; simon.lienardy@uliege.be;
5678908; Laurent Mathy; laurent.mathy@uliege.be; 2056
3217894; Benoit Donnet; benoit.donnet@uliege.be; 946
```

Insérez le corps de votre fonction dans le fichier `info0946-examen.c`. 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.C

Écrivez une fonction ou une procédure dont le nom est `statistiques` et qui retourne plusieurs statistiques sur les chambres d'hôte. Les statistiques qui nous intéressent sont le taux d'occupation actuel de vos chambres (i.e., nombre de chambres occupées / nombre total de chambres), le chiffre d'affaire actuel (durée d'occupation d'une chambre multipliée par le prix par nuitée de cette chambre) et, enfin, la durée moyenne d'une location. En plus du code, vous devez définir vous-même le prototype de la fonction ou procédure.

Insérez le prototype et le corps de votre fonction ou procédure `statistiques` dans le fichier `info0946-examen.c`. 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.D

Le fichier source `main-chambre.c` se présente comme suit :

```c
#include <stdio.h>
#include "chambre.h"

int main(){
  ChambresHotes *ch;
  float taux_occupation;
  float chiffre_affaire;
  float duree_moyenne;

  //1. Creation des chambres a partir d'un fichier Clients.txt, Chambres.txt,
  //pour 346 clients et 25 chambres.
  ...
  if(ch==NULL)
    return -1;

  //2. statistiques
  ...
  printf("Taux d'occupation des chambres: %f\n", taux_occupation);
  printf("Chiffre d'affaire actuel: %fEUR\n", chiffre_affaire);
  printf("Duree moyenne d'une location: %f\n", duree_moyenne);

  //3. sauve clients dans Clients.txt
  if(...){
    printf("Impossible de sauver les clients");
    return -1;
  }

  return 1;
}//fin programme
```

Indiquez, dans le fichier `info0946-examen.c`, les 3 lignes de code manquantes (c-à-d les " `...` ") à l'aide des fonctions/procédures définies tout au long de l'exercice. Attention, vous ne pouvez pas déclarer de variables supplémentaires.

Question 3.E

Indiquez, dans le fichier `info0946-examen.c`, la ligne de commande permettant de compiler votre code pour obtenir un fichier binaire exécutable appelé `hote`.