Correction

INF3135 **Examen Intra - Automne 2017**

Code Permanent :	Nom et prénom :
.....	.....

Consignes :

- **Durée : 3 heures.**
- La documentation est autorisée.
- Aucun appareil électronique n'est autorisé.
- Pour chaque question, **remplissez** au crayon noir ou bleu la case de la ou des bonnes réponses.
- Les questions faisant apparaître le symbole ♣ peuvent présenter zéro, une ou plusieurs bonnes réponses. Les autres ont une unique bonne réponse.
- En cas d'erreur sur une question à choix multiple (♣), la note à la question est 0. Pas de correction négative.
- Les réponses se font uniquement sur la page de formulaire (dernière page).
- À la fin de l'examen, vous devez remettre la page de formulaire (dernière page) contenant aussi vos réponses aux questions ouvertes. Vous pouvez conserver le reste du questionnaire pour référence lorsque vous recevrez la correction par courriel.

À moins qu'il en soit précisé autrement dans les questions, voici quelques détails sur l'architecture utilisée pour compiler les extraits de code C :

- Standard C11
- Taille d'un caractère (`char`) en mémoire : 1 octet
- Taille d'un entier (`int`) en mémoire : 4 octets
- Taille d'un booléen (`bool`) en mémoire : 1 octet
- Taille d'un pointeur (`void *`) en mémoire : 8 octets

Unix (6 points)

Soit la suite de commandes suivante :

```
1  mkdir tmp
2  touch tmp/a tmp/b tmp/c
3  mv tmp/a tmp/b
4  mkdir tmp2
5  cp tmp/* tmp2
6  rm -rf tmp
7  ls tmp2
```

Question 1 (3 points) Qu'affiche l'instruction `ls tmp2` à la ligne 7 ? On suppose que les répertoires `tmp` et `tmp2` n'existent pas avant l'exécution des commandes.

- [ ] A rien
- [x] b c
- [ ] C a
- [ ] D a c
- [ ] E c
- [ ] F b
- [ ] G a b
- [ ] H a b c

Question 2 (3 points) Le résultat de la commande `pwd` affiche `/home/user1/.vim/`. Quelle commande faut-il taper pour se rendre dans le répertoire `/home/user1/mon_projet/src` ?

- [ ] A cd /../mon_projet/src
- [ ] B cd mon_projet/src
- [x] cd ../mon_projet/src
- [ ] D cd /../../mon_projet/src
- [ ] E cd src
- [ ] F cd /mon_projet/src
- [ ] G cd ../../../mon_projet/src
- [ ] H cd ../../mon_projet/src
- [ ] I cd ./mon_projet/src

Git (10 points)

Le répertoire `mon_projet` contient les fichiers d'un projet que l'on souhaite versionner avec Git. La commande `ls -a` exécutée dans le répertoire `mon_projet` produit le résultat suivant :

```
.   fonctions.c   .git         main      .main.c.swp   Makefile      .README.md.swp
..  fonctions.o   .gitignore   main.c    main.o        README.md
```

Question 3 (2 points) Quelle commande faut-il taper pour initialiser le dépôt ?

- [ ] A git init
- [ ] B git add mon_projet
- [ ] C git commit -m "git_init"
- [x] aucune, le dépôt est déjà initialisé
- [ ] E git clone mon_projet
- [ ] F git pull
- [ ] G git log
- [ ] H git push

Question 4 ♣ (3 points) La commande **`git status`** produit le résultat ci-dessous :

```
On branch master

Initial commit

Untracked files:

        .README.md.swp
        .gitignore
        .main.c.swp
        Makefile
        README.md
        fonctions.c
        fonctions.o
        main.c
        main.o

nothing added to commit but untracked files present
```

Quelles sont les lignes à **ajouter** au fichier **`.gitignore`** ?

[A] mon_projet	[D] main	[G] main.c	[J] .git
[■] *.o	[E] *.md	[H] swp	[K] *.c
[C] .gitignore	[■] *.swp	[I] README.md	[L] fonctions.c

Question 5 (3 points) On souhaite maintenant sauvegarder (*commit*) les modifications faites seulement au fichier **`.gitignore`**. Quelle suite de commandes faut-il taper ?

[A] git add ∗
git commit −m "ignore_des_fichiers"

[B] git commit ∗
git push −m "ignore_des_fichiers"

[■] git add .gitignore
git commit −m "ignore_des_fichiers"

[D] git commit ∗
git add −m "ignore_des_fichiers"

[E] git add ∗
git push −m "ignore_des_fichiers"

[F] git commit −−add .gitignore
git add −m "ignore_des_fichiers"

Question 6 (2 points) À l'aide de toutes les informations dont vous disposez, qu'affiche la commande **`git log`** ?

[A] rien du tout	[C] une liste de 2 commits	[■] une liste de 1 commit
[B] que le dépôt est vide	[D] on ne peut pas savoir	[F] une liste de 3 commits

Makefile (10 points)

Le fichier Makefile ci-dessous permet de compiler le programme **`main`** qui utilise des fonctions définies dans le module **`fonctions`** :

```
1   .PHONY: # TODO 1
2
3   run: # TODO 2
4           ./prog
5
6   prog: # TODO 3
7           gcc -o prog prog.o
8
9   prog.o: # TODO 4
10          gcc -c prog.c
11
12  clean:
13          rm -f prog prog.o
```

Question 7 (2 points) Que faut-il écrire à la place de *# TODO 1* ?

A prog.c	D prog prog.o	■ run clean	J run clean prog
B prog.o	E rien	H prog.o prog.c	K prog prog.c
C clean	F run prog	I prog	L run

Question 8 (2 points) Que faut-il écrire à la place de *# TODO 2* ?

A clean	D prog prog.c	G run clean	J run
B prog.o prog.c	■ prog	H prog prog.o	K run clean prog
C run prog	F rien	I prog.c	L prog.o

Question 9 (2 points) Que faut-il écrire à la place de *# TODO 3* ?

■ prog.o	D prog prog.c	G prog prog.o	J run prog
B run clean prog	E prog.c	H rien	K clean
C prog.o prog.c	F run	I prog	L run clean

Question 10 (2 points) Que faut-il écrire à la place de *# TODO 4* ?

A run clean prog	D run clean	G rien	J prog prog.o
B run prog	E prog.o	H run	K clean
C prog	F prog prog.c	■ prog.c	L prog.o prog.c

Question 11 ♣ (2 points) Quels sont les fichiers produits lorsque l'on entre la commande **`make -B`** ? L'option **`-B`** permet de forcer la re-compilation de tous les fichiers.

A aucun	D gcc	G clean	■ prog.o
B prog.h	E Makefile	H run	K .PHONY
C rm	F TODO	I prog.c	■ prog

Langage C (74 points)

Tableaux statiques (6 points)

Soit le programme C suivant :

```
1  #include <stdio.h>
2
3  int main() {
4          int tab[10] = {1, 2, 3};
5
6          int s = sizeof(tab);
7          tab[2] = 4;
8          int length = /* TODO 1 */;
9
10         for(int i = 0; i < length; i++) {
11                 printf("%d ", /* TODO 2 */);
12         }
13
14         return 0;
15 }
```

Question 12 (2 points) Quelle est la valeur de `s` à la ligne 6 ?

☐ A 4	☐ C 12	☐ E 10	☐ G -1
■ 40	☐ D 11	☐ F 0	☐ H 3

Question 13 (2 points) Que faut-il écrire à la place de `/* TODO 1 */` à la ligne 8 pour calculer la longueur (c'est-à-dire le nombre d'éléments) du tableau `tab` ?

☐ A s	■ s / **sizeof**(**int**)	☐ E s − **sizeof**(**int**)
☐ B s ∗ **sizeof**(**int**)	☐ D **sizeof**(s)	☐ F **sizeof**(**int**)

Question 14 (2 points) Que faut-il écrire à la place de `/* TODO 2 */` à la ligne 11 pour afficher la valeur à l'indice `i` du tableau `tab` ?

☐ A tab[&i]	☐ C tab[∗i]	■ tab[i]
☐ B ∗tab[i]	☐ D ∗tab[&i]	☐ F &tab[i]

Pointeurs (8 points)

Soit le programme C suivant :

```
1  #include<stdio.h>
2
3  void increment(/* TODO 1 */) {
4          // Affiche la valeur actuelle de ‘i‘
5          /* TODO 2 */
6          // Incremente ‘i‘
7          ++*i;
8  }
9
10 int main() {
11         int i = 0;
12         // Apelle la fonction increment() avec ‘i‘
13         /* TODO 3 */
14         // Affiche la valeur actuelle de ‘i‘
15         printf("%d ", i);
16         // Affiche l'adresse de ‘i‘ en memoire
17         /* TODO 4 */
18         return 0;
19 }
```

Lorsqu'il est exécuté, le programme affiche le résultat suivant : 0 1 0x7ffd13d93324.

Question 15 (2 points) La fonction `increment()` permet d'incrémenter la valeur de l'entier `i` passé en paramètre. Que faut-il écrire comme signature à la fonction (*/* TODO 1 */*) à la ligne 3 ?

☐ A **const** ∗ **int** ∗i	☐ C **const** ∗ **int** i	☐ E **const int** i	☐ G **int** i
☐ B **int** ∗∗i	☐ D **const int** ∗i	☐ F **int** &i	■ **int** ∗i

Question 16 (2 points) On souhaite afficher la **valeur** de l'entier `i` sous forme décimale. Que faut-il écrire à la place de */* TODO 2 */* à la ligne 5 ?

■ printf("%d ", ∗i)	☐ C printf("%p ", i)	☐ E printf("%d ", i)	☐ G printf("%p ", &i)
☐ B printf(∗i)	☐ D printf("%p ", ∗i)	☐ F printf(i)	☐ H printf("%d ", &i)

Question 17 (2 points) On souhaite appeler la fonction `increment()` afin d'incrémenter le contenu de la variable `i`. Que faut-il écrire à la place de */* TODO 3 */* à la ligne 13 ?

☐ A increment(**const** ∗i)	☐ C increment(&&i)	☐ E increment(i)	☐ G increment(∗i)
■ increment(&i)	☐ D increment(∗∗i)	☐ F increment(i[0])	☐ H increment([i])

Question 18 (2 points) On souhaite afficher **l'adresse** de l'entier `i`. Que faut-il écrire à la place de */* TODO 4 */* à la ligne 17 ?

☐ A printf("%d ", ∗i)	☐ C printf("%s ", &i)	☐ E printf("%p ", ∗i)
☐ B printf("%p ", i)	☐ D printf("%d ", i)	■ printf("%p ", &i)

Arithmétique des pointeurs (8 points)

Soit le programme C suivant :

```
1  #include <stdio.h>
2
3  int main() {
4          int m[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
5          int *t[3] = { m[2], m[0], m[1] };
6          int **p = (int **) t;
7
8          printf("%d\n", *(t[1] + 2));
9          printf("%d\n", *(*p + 1));
10         printf("%d\n", *(*(p + 2) + 2));
11         printf("%d\n", **p);
12
13         return 0;
14 }
```

Question 19 (2 points) Qu'affiche l'instruction printf("%d", *(t[1] + 2)) à la ligne 8 ?

[A] 1 [B] 2 [■] 3 [D] 4 [E] 5 [F] 6 [G] 7 [H] 8 [I] 9

Question 20 (2 points) Qu'affiche l'instruction printf("%d", *(*p + 1)) à la ligne 9 ?

[A] 1 [B] 2 [C] 3 [D] 4 [E] 5 [F] 6 [G] 7 [■] 8 [I] 9

Question 21 (2 points) Qu'affiche l'instruction printf("%d", **p) à la ligne 11 ?

[A] 1 [B] 2 [C] 3 [D] 4 [E] 5 [F] 6 [■] 7 [H] 8 [I] 9

Unions (6 points)

Soit le programme C suivant :

```
1  #include <stdio.h>
2
3  union Entree {
4          char c;
5          int i;
6  };
7
8  int main() {
9          union Entree e;
10         e.c = 'A';
11         e.i = -10;
12         printf("%lu\n", sizeof(union Entree));
13         printf("%c\n", e.c);
14
15         return 0;
16 }
```

Question 22 (2 points) Qu'affiche printf("%lu", **sizeof**(**union** Entree)) à la ligne 12 ?

[A] -1 [■] 4 [E] NULL [G] 5
[B] 2 [D] 3 [F] 0 [H] 1

Question 23 (2 points) Sachant que `&e` vaut `0x7fff7c866060`, que vaut `&e.i` ?

A	0x7fff7c866065	C	0x7fff7c866064	E	0x7fff7c866061	G	0x7fff7c866066
B	0x7fff7c866067	D	0x7fff7c866063	F	0x7fff7c866062	■	0x7fff7c866060

Question 24 (2 points) Qu'affiche printf("%c", e.c) à la ligne 13 ?

A	0x7fff7c866061	C	A	E	NULL
B	-10	■	un caractère non spécifié	F	-1

Structures (6 points)

Soit le programme C suivant :

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  struct Etudiant {
5          char code[10];
6          bool estInscrit;
7  };
8
9  void inscrire(struct Etudiant e) {
10         e.estInscrit = true;
11 }
12
13 int main() {
14         printf("%lu\n", sizeof(struct Etudiant));
15
16         struct Etudiant e = { "DOEJ901130", false };
17
18         printf("%p\n", &e.estInscrit);
19
20         inscrire(e);
21
22         if(e.estInscrit) {
23                 printf("Inscrit\n");
24         } else {
25                 printf("Non inscrit\n");
26         }
27
28         return 0;
29 }
```

Question 25 (2 points) Qu'affiche printf("%lu", **sizeof**(**struct** Etudiant)) à la ligne 14 ?

A	NULL	C	-1	E	4	G	1
B	10	D	2	■	11	H	3

Question 26 (2 points) Sachant que `&e.code` vaut `0x7fff03dd8560`, qu'affiche printf("%p", &e.estInscrit) à la ligne 18 ?

A	0x7fff03dd8563	C	0x7fff03dd8562	E	0x7fff03dd856c	■	0x7fff03dd856a
B	0x7fff03dd856d	D	0x7fff03dd8560	F	0x7fff03dd856b	H	0x7fff03dd8561

Question 27 (2 points) Quelle est la dernière ligne affichée par le programme lorsqu'il est exécuté ?

■ Non inscrit B Inscrit C Segmentation fault

Allocation dynamique (10 points)

Soit le programme C suivant :

```
1  void fonction() {
2
3          int taille = 5;
4
5          // Alloue un tableau de longueur taille initialise a zero
6          int *tab = /* TODO 1 */;
7
8          // Copie les elements du tableau dans un nouveau tableau
9          int *copie = /* TODO 2 */;
10         for(int i = 0; i < taille; i++) {
11                 copie[i] = tab[i];
12         }
13
14         // Realloue le tableau pour l'agrandir
15         taille += 5;
16         tab = /* TODO 3 */;
17
18         printf("%lu\n", sizeof(tab));
19
20         // Affiche le contenu de la copie
21         for(int i = 0; i < taille; i++) {
22                 printf("%d ", copie[i]);
23         }
24         printf("\n");
25
26         free(copie);
27 }
```

Question 28 (2 points) Que faut-il écrire à la place de `/* TODO 1 */` à la ligne 6 pour **allouer et initialiser à 0** le tableau ?

A calloc(taille * **sizeof**(**int**))
B malloc(taille * **sizeof**(**int**))
C realloc(tab, taille , **sizeof**(**int**))
D malloc(**sizeof**(**int**))
E realloc(tab, taille * **sizeof**(**int**))
F malloc(taille , **sizeof**(**int**))
G calloc(taille)
■ calloc(taille , **sizeof**(**int**))

Question 29 (2 points) Que faut-il écrire à la place de `/* TODO 2 */` à la ligne 9 pour obtenir une initialisation qui ne met pas toutes les valeurs à 0 ?

■ malloc(taille * **sizeof**(**int**))
B calloc(taille)
C malloc(**sizeof**(**int**))
D malloc(taille , **sizeof**(**int**))
E calloc(taille * **sizeof**(**int**))
F realloc(tab, taille , **sizeof**(**int**))
G calloc(taille , **sizeof**(**int**))
H realloc(tab, taille * **sizeof**(**int**))

Question 30 (2 points) Que faut-il écrire à la place de `/* TODO 3 */` à la ligne 16 pour agrandir l'espace alloué au tableau ?

- [A] malloc(taille , **sizeof**(**int**))
- [B] malloc(taille ∗ **sizeof**(**int**))
- [C] malloc(**sizeof**(**int**))
- [D] calloc (taille)
- [E] realloc (tab, taille , **sizeof**(**int**))
- [F] calloc (taille ∗ **sizeof**(**int**))
- [■] realloc (tab, taille ∗ **sizeof**(**int**))
- [H] calloc (taille , **sizeof**(**int**))

Question 31 (2 points) Qu'affiche l'instruction printf ("%lu", **sizeof**(tab)) à la ligne 18 ?

- [A] 10
- [■] 8
- [C] 20
- [D] 5
- [E] 0
- [F] 40

Question 32 (2 points) Que pouvez-vous dire au sujet de la gestion de la mémoire faite par la fonction ?

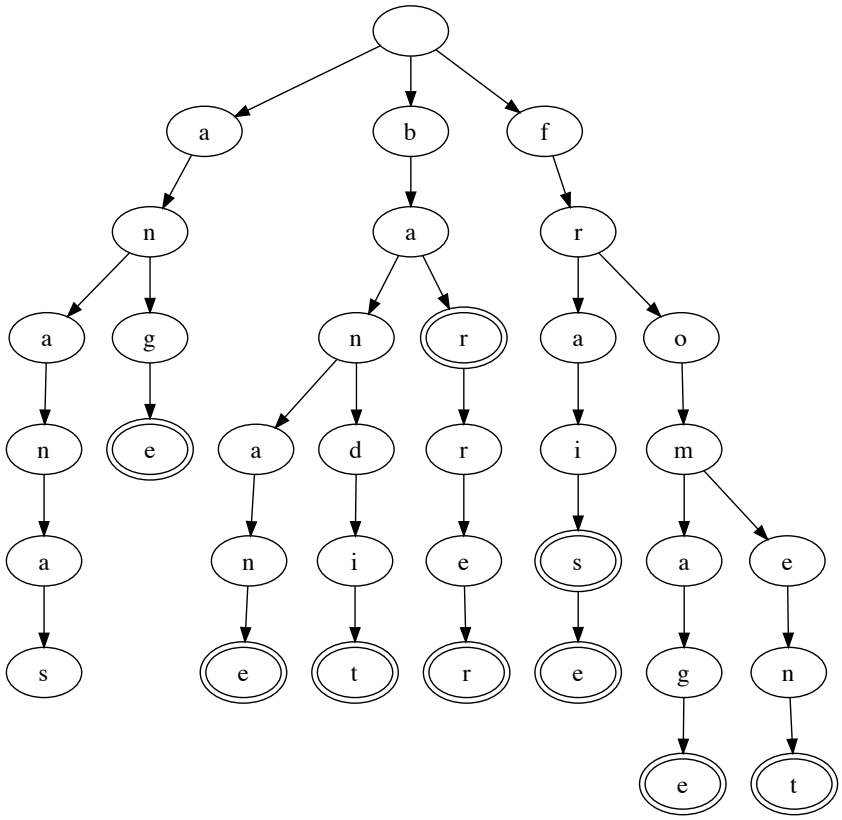
- [A] Elle provoque une erreur de segmentation
- [B] Elle ne provoque pas de fuite de mémoire
- [■] Elle provoque une fuite de mémoire

Structures de données (30 points)

Une structure de données fondamentale en informatique est le *dictionnaire*, qui est essentiellement une collection de chaînes de caractères sur laquelle on effectue les opérations suivantes :

1. *Créer* un dictionnaire vide ;
2. *Afficher* le contenu du dictionnaire en ordre alphabétique ;
3. *Insérer* un mot dans le dictionnaire ;
4. *Supprimer* un mot d'un dictionnaire ;
5. *Détruire* un dictionnaire.

Il existe plusieurs façons d'implémenter un *dictionnaire* : à l'aide d'un tableau redimensionnable, à l'aide d'un arbre binaire, etc. Dans cette question, nous nous intéressons à une implémentation appelée *arbre préfixe*, qui consiste à représenter les mots insérés à l'aide d'un arbre. Voici un exemple d'un dictionnaire implémenté à l'aide d'un arbre préfixe qui contient les mots *ange*, *banane*, *bandit*, *bar*, *barrer*, *frais*, *fraise*, *fromage* et *froment* : il suffit de lire à partir du haut vers le bas en suivant les lettres, une double ellipse indiquant que le mot se trouve dans le dictionnaire.



Vous devez compléter le fichier C suivant qui implémente la structure de données d'arbre préfixe. Les fonctions incomplètes sont les suivantes :

— Dictionnaire_creer (identifié par */* TODO 1 */*) ;
— Dictionnaire_contientMotRecursif (identifié par */* TODO 2 */*) ;
— Dictionnaire_detruireRecursif (identifié par */* TODO 3 */*).

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NB_LETTRES  26
#define LNG_MAX_MOT 30

// ---------------------- //
// Structures de donnees //
// ---------------------- //

struct Noeud {                          // Un noeud du dictionnaire
    bool terminal;                      // Vrai si le noeud correspond a un mot
    struct Noeud *enfants[NB_LETTRES]; // Un tableau de pointeurs de noeuds
};

struct Dictionnaire {     // Un dictionnaire
    struct Noeud *racine; // La racine du dictionnaire
    unsigned int nbMots;  // Le nombre de mots dans le dictionnaire
};

// ---------- //
// Prototypes //
// ---------- //

/**
 * Retourne un dictionnaire vide.
 *
 * @return  Le dictionnaire
 */
struct Dictionnaire Dictionnaire_creer();

/**
 * Affiche le contenu d'un dictionnaire sur stdout.
 *
 * @return  Le dictionnaire a afficher
 */
void Dictionnaire_afficher(const struct Dictionnaire *dictionnaire);

/**
 * Insere un mot dans un dictionnaire.
 *
 * Si le mot est deja present, ne fait rien.
 *
 * @param dictionnaire  Le dictionnaire dans lequel on insere
 * @param mot           Le mot a inserer
 */
void Dictionnaire_insererMot(struct Dictionnaire *dictionnaire, const char *mot);

/**
 * Indique si un mot se trouve dans un dictionnaire
 *
 * @param dictionnaire  Le dictionnaire a consulter
 * @param mot           Le mot a verifier
 * @return              Vrai si et seulement si mot est dans dictionnaire
 */
bool Dictionnaire_contientMot(const struct Dictionnaire *dictionnaire,
                              const char *mot);

/**
 * Supprime un mot d'un dictionnaire.
 *
 * Si le mot n'est pas present, ne fait rien.
 *
 * @param dictionnaire  Le dictionnaire
 * @param mot           Le mot a supprimer
 */
void Dictionnaire_supprimerMot(struct Dictionnaire *dictionnaire,
                               const char *mot);

/**
 * Detruit un dictionnaire.
 *
 * @param dictionnaire  Le dictionnaire a detruire
 */
void Dictionnaire_detruire(struct Dictionnaire *dictionnaire);

// --------------- //
// Implementation //
// --------------- //
```

```
82
83  struct Dictionnaire Dictionnaire_creer() {
84      /**********/
85      /* TODO 1 */
86      /**********/
87  }
88
89  /**
90   * Affiche recursivement le contenu d'un noeud.
91   *
92   * @param noeud       Le noeud a partir duquel on fait l'affichage
93   * @param prefixe     Le mot correpondant au noeud courant
94   * @param profondeur  La profondeur du noeud courant
95   */
96  void Dictionnaire_afficherRecursif(const struct Noeud *noeud,
97                                     char prefixe[LNG_MAX_MOT],
98                                     unsigned int profondeur) {
99      if (noeud != NULL) {
100         if (noeud->terminal) printf("%s\n", prefixe);
101         for (unsigned int i = 0; i < NB_LETTRES; ++i) {
102             prefixe[profondeur] = 'a' + i;
103             prefixe[profondeur + 1] = '\0';
104             Dictionnaire_afficherRecursif(noeud->enfants[i],
105                                           prefixe, profondeur + 1);
106         }
107     }
108 }
109
110 void Dictionnaire_afficher(const struct Dictionnaire *dictionnaire) {
111     char prefixe[LNG_MAX_MOT + 1] = "";
112     Dictionnaire_afficherRecursif(dictionnaire->racine, prefixe, 0);
113 }
114
115 /**
116  * Insere recursivement un mot dans un dictionnaire.
117  *
118  * @param noeud    Le noeud courant
119  * @param suffixe  La portion de mot pas encore lue
120  */
121 void Dictionnaire_insererMotRecursif(struct Noeud **noeud,
122                                      const char *suffixe) {
123     if (*noeud == NULL) {
124         *noeud = malloc(sizeof(struct Noeud));
125         (*noeud)->terminal = false;
126         for (unsigned int i = 0; i < NB_LETTRES; ++i) {
127             (*noeud)->enfants[i] = NULL;
128         }
129     }
130     if (strcmp(suffixe, "") == 0) {
131         (*noeud)->terminal = true;
132     } else {
133         unsigned int i = suffixe[0] - 'a';
134         Dictionnaire_insererMotRecursif(&(*noeud)->enfants[i], suffixe + 1);
135     }
136 }
137
138 void Dictionnaire_insererMot(struct Dictionnaire *dictionnaire, const char *mot) {
139     Dictionnaire_insererMotRecursif(&dictionnaire->racine, mot);
140 }
141
142 /**
143  * Verifie recursivement la presence d'un mot dans un dictionnaire.
144  *
145  * @param noeud    Le noeud courant
146  * @param suffixe  La portion de mot pas encore lue
147  */
148 bool Dictionnaire_contientMotRecursif(const struct Noeud *noeud,
149                                       const char *suffixe) {
150     /**********/
151     /* TODO 2 */
152     /**********/
153 }
154
155 bool Dictionnaire_contientMot(const struct Dictionnaire *dictionnaire,
156                               const char *mot) {
157     return Dictionnaire_contientMotRecursif(dictionnaire->racine, mot);
158 }
159
160 /**
161  * Supprime recursivement un mot d'un dictionnaire.
162  *
```

```c
 * @param noeud     Le noeud courant
 * @param suffixe  La portion de mot pas encore lue
 */
void Dictionnaire_supprimerMotRecursif(struct Noeud *noeud,
                                       const char *suffixe) {
    if (noeud != NULL) {
        if (strcmp(suffixe, "") == 0) {
            noeud->terminal = false;
        } else {
            unsigned int i = suffixe[0] - 'a';
            Dictionnaire_supprimerMotRecursif(noeud->enfants[i],
                                              suffixe + 1);
        }
    }
}

void Dictionnaire_supprimerMot(struct Dictionnaire *dictionnaire, const char *mot) {
    Dictionnaire_supprimerMotRecursif(dictionnaire->racine, mot);
}

/**
 * Detruit recursivement un dictionnaire.
 *
 * @param noeud  Le noeud courant
 */
void Dictionnaire_detruireRecursif(struct Noeud *noeud) {
    /**********/
    /* TODO 3 */
    /**********/
}

void Dictionnaire_detruire(struct Dictionnaire *dictionnaire) {
    Dictionnaire_detruireRecursif(dictionnaire->racine);
}

// ---- //
// Main //
// ---- //

int main() {
    struct Dictionnaire dictionnaire;
    dictionnaire = Dictionnaire_creer();
    Dictionnaire_insererMot(&dictionnaire, "frais");
    Dictionnaire_insererMot(&dictionnaire, "banane");
    Dictionnaire_insererMot(&dictionnaire, "froment");
    Dictionnaire_insererMot(&dictionnaire, "bandit");
    Dictionnaire_insererMot(&dictionnaire, "bar");
    Dictionnaire_insererMot(&dictionnaire, "fromage");
    Dictionnaire_insererMot(&dictionnaire, "ananas");
    Dictionnaire_insererMot(&dictionnaire, "barrer");
    Dictionnaire_insererMot(&dictionnaire, "fraise");
    Dictionnaire_insererMot(&dictionnaire, "bar");
    Dictionnaire_insererMot(&dictionnaire, "ange");
    Dictionnaire_insererMot(&dictionnaire, "froment");
    Dictionnaire_supprimerMot(&dictionnaire, "ananas");
    Dictionnaire_afficher(&dictionnaire);
    printf("\"ananas\" dans dictionnaire ? %s\n",
           Dictionnaire_contientMot(&dictionnaire, "ananas") ?
           "oui" : "non");
    printf("\"tomate\" dans dictionnaire ? %s\n",
           Dictionnaire_contientMot(&dictionnaire, "tomate") ?
           "oui" : "non");
    printf("\"anana\" dans dictionnaire ? %s\n",
           Dictionnaire_contientMot(&dictionnaire, "anana") ?
           "oui" : "non");
    Dictionnaire_detruire(&dictionnaire);
    return 0;
}
```

Question 33 (5 points) Que faut-il écrire à la place de `/* TODO 1 */` pour implémenter la fonction `Dictionnaire_creer` ?

Question 34 (15 points) Que faut-il écrire à la place de `/* TODO 2 */` pour implémenter la fonction `Dictionnaire_contientMotRecursif` ?

Question 35 (10 points) Que faut-il écrire à la place de `/* TODO 3 */` pour implémenter la fonction `Dictionnaire_detruireRecursif` ?

Correction

INF3135 **Examen Intra - Automne 2017**

Code Permanent :	Nom et prénom :
.....	.....

Question 1 : [A] [■] [C] [D] [E] [F] [G] [H]
Question 2 : [A] [B] [■] [D] [E] [F] [G] [H] [I]
Question 3 : [A] [B] [C] [■] [E] [F] [G] [H]
Question 4 : [A] [■] [C] [D] [E] [■] [G] [H] [I] [J] [K] [L]
Question 5 : [A] [B] [■] [D] [E] [F]
Question 6 : [A] [B] [C] [D] [■] [F]
Question 7 : [A] [B] [C] [D] [E] [F] [■] [H] [I] [J] [K] [L]
Question 8 : [A] [B] [C] [D] [■] [F] [G] [H] [I] [J] [K] [L]
Question 9 : [■] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L]
Question 10 : [A] [B] [C] [D] [E] [F] [G] [H] [■] [J] [K] [L]
Question 11 : [A] [B] [C] [D] [E] [F] [G] [H] [I] [■] [K] [■]
Question 12 : [A] [■] [C] [D] [E] [F] [G] [H]
Question 13 : [A] [B] [■] [D] [E] [F]
Question 14 : [A] [B] [C] [D] [■] [F]
Question 15 : [A] [B] [C] [D] [E] [F] [G] [■]
Question 16 : [■] [B] [C] [D] [E] [F] [G] [H]
Question 17 : [A] [■] [C] [D] [E] [F] [G] [H]
Question 18 : [A] [B] [C] [D] [E] [■]
Question 19 : [A] [B] [■] [D] [E] [F] [G] [H] [I]
Question 20 : [A] [B] [C] [D] [E] [F] [G] [■] [I]
Question 21 : [A] [B] [C] [D] [E] [F] [■] [H] [I]
Question 22 : [A] [B] [■] [D] [E] [F] [G] [H]
Question 23 : [A] [B] [C] [D] [E] [F] [G] [■]
Question 24 : [A] [B] [C] [■] [E] [F]
Question 25 : [A] [B] [C] [D] [E] [■] [G] [H]
Question 26 : [A] [B] [C] [D] [E] [F] [■] [H]
Question 27 : [■] [B] [C]
Question 28 : [A] [B] [C] [D] [E] [F] [G] [■]
Question 29 : [■] [B] [C] [D] [E] [F] [G] [H]
Question 30 : [A] [B] [C] [D] [E] [F] [■] [H]
Question 31 : [A] [■] [C] [D] [E] [F]
Question 32 : [A] [B] [■]
Question 33 :
Question 34 :
Question 35 :