Correction

INF3135 **Examen Final - 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

Généricité et pointeurs de fonctions (15 points)

Considérez le programme (incomplet) suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Date {
5      unsigned int jour;
6      unsigned int mois;
7      unsigned int annee;
8  };
9
10 void afficherDate(const struct Date *d) {
11     printf("%d/%d/%d", d->jour, d->mois, d->annee);
12 }
13
14 int comparerDates(const void *d1, const void *d2) {
15     const struct Date *date1, *date2;
16     date1 = (const struct Date*)d1;
17     date2 = (const struct Date*)d2;
18     if (date1->annee == date2->annee) {
19         if (date1->mois == date2->mois) {
20             return date1->jour - date2->jour;
21         } else {
22             return date1->mois - date2->mois;
23         }
24     } else {
25         return date1->annee - date2->annee;
26     }
27 }
28
29 int main() {
30     struct Date dates[5] = {
31         {.jour =  8, .mois =  9, .annee = 2001},
32         {.jour = 28, .mois =  5, .annee = 1971},
33         {.jour = 11, .mois =  2, .annee = 2001},
34         {.jour = 31, .mois = 12, .annee = 0},
35         {.jour = 14, .mois =  3, .annee = 855}
36     };
37     printf("Avant tri\n----------\n");
38     for (unsigned int i = 0; i < 5; ++i) {
39         afficherDate(&dates[i]); printf("\n");
40     }
41     qsort(/* TODO 1 */,
42           /* TODO 2 */,
43           /* TODO 3 */,
44           /* TODO 4 */
45     );
46     printf("\nApres tri\n----------\n");
47     for (unsigned int i = 0; i < 5; ++i) {
48         afficherDate(&dates[i]); printf("\n");
49     }
50     printf("\nLecture d'une date sur stdin...\n");
51     struct Date date;
```

```
52      scanf("%d %d %d", &date.jour, &date.mois, &date.annee);
53      printf("Est-ce que la date "); afficherDate(&date);
54      printf(" se trouve dans le tableau? ");
55      if (bsearch(/* TODO 5 */,
56                  /* TODO 6 */,
57                  /* TODO 7 */,
58                  /* TODO 8 */,
59                  /* TODO 9 */
60          ) == NULL) {
61          printf("non\n");
62      } else {
63          printf("oui\n");
64      }
65      return 0;
66  }
```

Lorsqu'on l'exécute, on obtient le résultat suivant sur la sortie standard :

```
1   Avant tri
2   ---------
3   8/9/2001
4   28/5/1971
5   11/2/2001
6   31/12/0
7   14/3/855
8
9   Apres tri
10  ---------
11  31/12/0
12  14/3/855
13  28/5/1971
14  11/2/2001
15  8/9/2001
16
17  Lecture d'une date sur stdin...
18  <L'utilisateur saisit par exemple 16 12 2017>
19  Est-ce que la date 16/12/2017 se trouve dans le tableau? non
```

Question 1 (2 points) Quel est le 1er argument à passer à la fonction `qsort` (`/* TODO 1 */`) ?

☐ A *dates ☐ B &dates ☐ C dates[0] ☐ D **dates ■ dates

Question 2 (2 points) Quel est le 2e argument à passer à la fonction `qsort` (`/* TODO 2 */`) ?

☐ A **sizeof**(**struct** Date) ☐ C 3 ☐ E **sizeof**(**int**)

☐ B NULL ☐ D **sizeof**(Date) ■ 5

Question 3 (2 points) Quel est le 3e argument à passer à la fonction `qsort` (`/* TODO 3 */`) ?

☐ A **sizeof**(Date) ☐ C NULL ■ **sizeof**(**struct** Date)

☐ B 3 ☐ D **sizeof**(**int**) ☐ F 5

Question 4 (2 points) Quel est le 4e argument à passer à la fonction `qsort` (`/* TODO 4 */`) ?

- [A] &comparerDates
- [C] ∗comparerDates
- [■] comparerDates
- [B] ∗∗comparerDates
- [D] NULL
- [F] &&comparerDates

Question 5 (2 points) Quel est le 1er argument à passer à la fonction `bsearch` (`/* TODO 5 */`) ?

- [A] date
- [C] ∗date
- [■] &date
- [B] ∗∗date
- [D] NULL
- [F] &&date

Question 6 (2 points) Quel est le 2e argument à passer à la fonction `bsearch` (`/* TODO 6 */`) ?

- [■] dates
- [C] ∗date
- [E] ∗dates
- [B] date
- [D] &dates
- [F] ∗∗dates

Question 7 (1 point) Quel est le 3e argument à passer à la fonction `bsearch` (`/* TODO 7 */`) ?

- [A] 1
- [C] **sizeof**(**struct** Date)
- [E] NULL
- [■] 5
- [D] 3
- [F] **sizeof**(**int**)

Question 8 (1 point) Quel est le 4e argument à passer à la fonction `bsearch` (`/* TODO 8 */`) ?

- [A] 5
- [C] **sizeof**(Date)
- [E] **sizeof**(**int**)
- [B] NULL
- [■] **sizeof**(**struct** Date)
- [F] 3

Question 9 (1 point) Quel est le 5e argument à passer à la fonction `bsearch` (`/* TODO 9 */`) ?

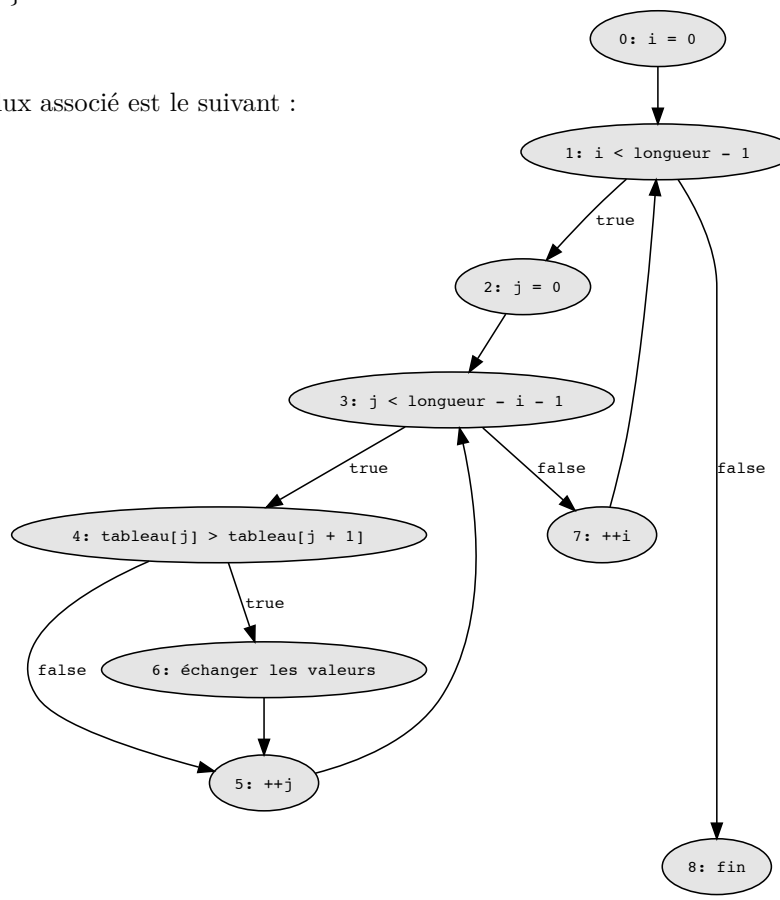
- [A] NULL
- [■] comparerDates
- [E] &comparerDates
- [B] &&comparerDates
- [D] ∗∗comparerDates
- [F] ∗comparerDates

Graphe de flux et tests (15 points)

Considérez la fonction C suivante qui effectue un tri (*bubble sort*).

```
1  void bubbleSort(int tableau[], unsigned int longueur) {
2      for (unsigned int i = 0; i < longueur - 1; ++i) {
3          for (unsigned int j = 0; j < longueur - i - 1; ++j) {
4              if (tableau[j] > tableau[j + 1]) {
5                  int temp = tableau[j];
6                  tableau[j] = tableau[j + 1];
7                  tableau[j + 1] = temp;
8              }
9          }
10     }
11 }
```

Son graphe de flux associé est le suivant :



Question 10 (3 points) Quelle est la complexité cyclomatique de cette fonction ?

A 5 ■ 4 C 6 D 2 E 3 F 1

Question 11 (3 points) Parmi les jeux de valeurs suivants pour `tableau` et `longueur`, lequel permet de réaliser le chemin 0-1-8 (les numéros indiquent les sommets visités dans le graphe de flux) ?

■ `tableau = {3,1,2}`, `longueur = 1`

C `tableau = {1,2,3}`, `longueur = 3`

B Aucun des jeux proposés

D `tableau = {2,1,3}`, `longueur = 3`

Question 12 (3 points) Supposons que la fonction `bubbleSort` soit appelée avec les arguments `tableau = {2,1}` et `longueur = 2`. Quel sera le chemin parcouru dans le graphe de flux ? *Remarque :* Pour avoir une écriture plus compacte, nous utilisons la notation exponentielle, pour indiquer une répétition. Par exemple, le chemin 1-2-3-4-5-3-4-5 s'écrit 1-2-(3-4-5)^2.

- [■] 0-1-2-3-4-6-5-3-7-1-8
- [B] 0-1-2-3-(4-6-5-3)^2-7-1-8
- [C] 0-1-2-3-7-1-8
- [D] 0-1-8
- [E] Aucun chemin parmi ceux proposés
- [F] 0-1-2-3-4-5-3-7-1-8

Question 13 (3 points) Même question, si `tableau = {3,1,2}` et `longueur = 3`.

- [A] 0-1-2-3-4-5-3-7-1-8
- [B] 0-(1-2-3-(4-5-3)^2-7)^2-1-8
- [C] 0-1-8
- [■] 0-1-2-3-(4-5-3)^2-7-1-2-3-4-5-3-7-1-8
- [E] 0-1-2-3-7-1-8
- [F] Aucun chemin parmi ceux proposés

Question 14 ♣ (3 points) Bob propose de tester la fonction `bubbleSort` avec le jeu de 5 tests décrits dans le tableau suivant :

`tableau`	`longueur`
{}	`0`
{`3`}	`1`
{`1,2,3`}	`3`
{`3,2,1`}	`3`
{`1,2,3,4`}	`4`

Indiquez toutes les affirmations qui sont vraies à propos de ce cadre de tests parmi les suivantes :

- [A] Les chemins associés aux tests sont indépendants (couverture non redondante)
- [B] Ce cadre met en évidence un bogue de la fonction
- [C] Ce cadre garantit qu'il n'y a aucun bogue dans la fonction
- [■] Ce cadre couvre tous les branchements (flèches du graphe de flux)

Scripts Shell (20 points)

Question 15 (2 points) On souhaite ajouter la ligne `"*.o"` au fichier `.gitignore` à l'aide d'une commande Shell et ce, sans ouvrir le fichier ni supprimer son contenu.

Quelle commande doit-on taper ?

- [A] `ls "*.o" > .gitignore`
- [B] `cat "*.o" >> .gitignore`
- [C] `echo "*.o" > .gitignore`
- [D] `cat "*.o" > .gitignore`
- [■] `echo "*.o" >> .gitignore`
- [F] `ls "*.o" >> .gitignore`

Question 16 (3 points) Soit le script Shell suivant appelé **mon_script2.sh** :

```
1  #!/bin/bash
2  echo ``Hello, World!"
```

Et voici la sortie affichée par la commande **ls -lha** :

```
-rw-rw-r-- 1 user user 852 Dec 01 12:00 mon_script2.sh
```

On souhaite exécuter le script **mon_script2.sh**. Quelle commande doit-on taper ?

- [A] `gcc ./mon_script2.sh`
- [B] `./mon_script2.sh`
- [C] `cat ./mon_script2.sh`
- [D] `ls ./mon_script2.sh`
- [E] `echo ./mon_script2.sh`
- [■] `/bin/bash mon_script2.sh`

Question 17 (2 points) Le projet C **mon_projet** contient un fichier Makefile permettant de compiler l'exécutable **mon_programme** depuis les sources C.
On souhaite compiler puis exécuter le programme en une seule ligne. Bien sûr, on ne souhaite lancer l'exécution de **mon_programme** que si l'exécution du **make** s'est bien déroulée.

Quelle commande faut-il taper ?

- [A] `make & ./mon_programme`
- [B] `./mon_programme || make`
- [C] `./mon_programme | make`
- [D] `make; ./mon_programme`
- [E] `make | ./mon_programme`
- [F] `./mon_programme && make`
- [■] `make && ./mon_programme`
- [H] `./mon_programme & make`
- [I] `./mon_programme; make`
- [J] `make || ./mon_programme`

Le script suivant est une ébauche permettant de tester notre programme.

```
1  #!/bin/bash
2
3  make clean
4  make -B
5
6  if # TODO 1
7  then
8          echo "Erreur de compilation"
9          exit 1
10 fi
11
12 if # TODO 2
13 then
14         echo "Le programme généré ne s'apelle pas \`mon_programme\`"
15         exit 1
16 fi
17
18 echo "Compilation OK"
```

Question 18 (3 points) Dans un premier temps on souhaite s'assurer que le Makefile appelé ne provoque pas d'erreur.

Que faut-il écrire à la place de `# TODO 1` ?

☐ A **test** $$ -le 1	☐ E **test** $$ -eq 1	☐ I **test** $$ -eq 0
☐ B **test** $? -n	☐ F **test** $? -le 1	☐ J **test** $? -z
☐ C **test** $? -eq 1	☐ G **test** $? -eq 0	☐ K **test** $? -le 0
☐ D **test** $$ -le 0	■ **test** $? -ne 0	☐ L **test** $$ -ne 0

Question 19 (3 points) On souhaite maintenant s'assurer que le Makefile génère bien un fichier exécutable nommé **mon_programme**.

Que faut-il écrire à la place de `# TODO 2` ?

☐ A **test** -f mon_programme	☐ E **test** ! -e mon_programme	☐ I **test** ! -w mon_programme
☐ B **test** -x mon_programme	☐ F **test** ! -r mon_programme	☐ J **test** ! -d mon_programme
☐ C **test** -d mon_programme	☐ G **test** -r mon_programme	☐ K **test** -w mon_programme
■ **test** ! -x mon_programme	☐ H **test** -e mon_programme	☐ L **test** ! -f mon_programme

Question 20 (3 points) La commande **`git log --pretty=oneline`** affiche le résultat suivant :

```
67c575 README: corrige une typo
3693ce prog.h: ajoute les DocString
9d90c1 prog.c: vérifie les arguments
37f0a4 README: documente l'usage du programme
5794af prog.c: ajoute l'implémentation
bf3187 prog.h: ajoute les déclarations
e2c0f6 git init
```

Qu'affiche la commande suivante ?

```
git log --pretty=oneline | grep prog.c | head -n 1 | sed "s/.*: //"
```

■ "vérifie les arguments"	D "" (chaîne vide)	G "corrige une typo"
B "5794af"	E "README"	H "9d90c1"
C "ajoute l'implémentation"	F "67c575"	I "prog.c"

Soit le script **`fonction.sh`** suivant :

```
1  #!/bin/bash
2
3  fonction()
4  {
5          echo $1;
6          shift 2;
7          echo $1;
8  }
9
10 fonction $0 $1 $#
```

On exécute le script avec la commande suivante :

```
./fonction.sh foo bar baz
```

Question 21 (2 points) Qu'affiche le programme à la **`ligne 5`** ?

A "foo"	D "3"	G "foo bar baz"	J "2"
B "baz"	E Rien du tout	H "fonction"	K "123"
C "bar"	F "1"	■ "./fonction.sh"	L "$1"

Question 22 (2 points) Qu'affiche le programme à la **`ligne 7`** ?

A Rien du tout	D "bar"	G "baz"	J "2"
B "fonction"	E "123"	■ "3"	K "$1"
C "foo bar baz"	F "foo"	I "./fonction.sh"	L "1"

Git (10 points)

Question 23 (2 points) La commande `git remote -v` affiche le résultat suivant :

```
1  origin  git@gitlab.com:User1/mon_projet.git (fetch)
2  origin  git@gitlab.com:User1/mon_projet.git (push)
3  upstream  git@gitlab.com:User2/mon_projet.git (fetch)
4  upstream  git@gitlab.com:User2/mon_projet.git (push)
```

On souhaite modifier le contenu du répertoire contenant le projet pour qu'il corresponde à celui de la branche `ma_branche` disponible sur le dépôt de l'utilisateur `User2`.
Quelle commande faut-il taper ?

- [ ] A `git checkout origin/ma_branche`
- [ ] B `git pull upstream/ma_branche`
- [ ] C `git pull origin/ma_branche`
- [ ] D `git merge upstream/ma_branche`
- [ ] E `git merge origin/ma_branche`
- [x] `git checkout upstream/ma_branche`

Question 24 (2 points) La commande `git branch` affiche le résultat suivant :

```
1  master
2  * ma_branche
3  branche1
4  branche2
```

Actuellement, la branche `branche2` est basée sur la branche `master`. On souhaite la modifier pour la baser sur la branche `branche1`. C'est à dire que dans la branche `branche2`, les commits de `branche2` seront après les commits de la branche `branche1`.

Quelles commandes faut-il taper ?

- [ ] A `git rebase branche2`
- [ ] B `git rebase branche1`
- [ ] C
```
git checkout branche2
git merge branche1
```
- [x]
```
git checkout branche2
git rebase branche1
```
- [ ] E
```
git checkout branche1
git merge branche2
```
- [ ] F
```
git checkout branche1
git rebase branche2
```

Question 25 (3 points) La commande `git log --pretty=oneline` affiche le résultat suivant :

```
67c575 README: corrige une typo
3693ce prog.h: ajoute les DocString
9d90c1 prog.c: vérifie les arguments
37f0a4 README: documente l'usage du programme
5794af prog.c: ajoute l'implémentation
bf3187 prog.h: ajoute les déclarations
e2c0f6 git init
```

On souhaite utiliser le *rebasage interactif* de Git pour fusionner les commits `37f0a4` et `67c575`.

Sachant que `HEAD~k` désigne le $k-$ième parent du *commit* courant, quelle commande faut-il taper ?

- ☐ A `git rebase HEAD~4`
- ☐ B `git rebase 67c575`
- ☐ C `git rebase -i 9d90c1`
- ■ `git rebase -i HEAD~4`
- ☐ E `git rebase -i HEAD~1`
- ☐ F `git rebase -i 67c575`
- ☐ G `git rebase HEAD~1`
- ☐ H `git rebase 9d90c1`

Question 26 (3 points) La commande de rebasage de la question précédente ouvre Vim (ou n'importe quel éditeur de texte que vous avez configuré) et affiche le contenu suivant :

```
pick 37f0a4 README: documente l'usage du programme
pick 9d90c1 prog.c: vérifie les arguments
pick 3693ce prog.h: ajoute les DocString
pick 67c575 README: corrige une typo
```

Quelles modifications faut-il apporter au fichier pour fusionner les commits `37f0a4` et `67c575` en un seul commit ? On ne souhaite pas conserver le message du commit `67c575`.

- ☐ A Changer le `pick` de la ligne 4 par `edit` et déplacer la ligne juste **après la ligne 1**
- ☐ B Changer le `pick` de la ligne 4 par `squash` et déplacer la ligne juste **avant la ligne 1**
- ☐ C Changer le `pick` de la ligne 4 par `squash` et déplacer la ligne juste **après la ligne 1**
- ☐ D Changer le `pick` de la ligne 4 par `edit` et déplacer la ligne juste **avant la ligne 1**
- ☐ E Changer le `pick` de la ligne 4 par `fixup` et déplacer la ligne juste **avant la ligne 1**
- ■ Changer le `pick` de la ligne 4 par `fixup` et déplacer la ligne juste **après la ligne 1**

Détection de fuites mémoire (10 points)

On exécute la commande Valgrind sur le programme **mon_programme**.

La commande produit le résultat suivant :

```
1  ==5503== Memcheck, a memory error detector
2  ==5503== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
3  ==5503== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
4  ==5503== Command: ./mon_programme --help
5  ==5503==
6  Usage: ./mon_programme [--help]
7
8  Mon exemple de programme.
9  ==5503==
10 ==5503== HEAP SUMMARY:
11 ==5503==     in use at exit: 0 bytes in 0 blocks
12 ==5503==   total heap usage: 31 allocs, 31 frees, 4,705 bytes allocated
13 ==5503==
14 ==5503== All heap blocks were freed -- no leaks are possible
15 ==5503==
16 ==5503== For counts of detected and suppressed errors, rerun with: -v
17 ==5503== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Question 27 (3 points) En vous fiant uniquement à la sortie Valgrind, que pouvez-vous déduire sur le programme **mon_programme** ?

■ Le programme avec l'option **--help** ne provoque pas de fuite mémoire.

B Le programme provoque une fuite mémoire.

C Le programme provoque une erreur de segmentation

D Le programme avec l'option **--help** provoque une erreur de segmentation

E Le programme ne provoque pas de fuite mémoire.

F Le programme avec l'option **--help** provoque une fuite mémoire.

Question 28 (3 points) Supposons maintenant que vous exécutiez Valgrind sur le même programme en lui passant un paramètre `foo` et que vous obteniez le résultat suivant :

```
1  ==883== Memcheck, a memory error detector
2  ==883== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
3  ==883== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
4  ==883== Command: ./mon_programme foo
5  ==883==
6  Hello, foo!
7  ==883==
8  ==883== HEAP SUMMARY:
9  ==883==     in use at exit: 13,262 bytes in 28 blocks
10 ==883==   total heap usage: 51 allocs, 23 frees, 15,294 bytes allocated
11 ==883==
12 ==883== 40 bytes in 1 blocks are definitely lost in loss record 4 of 15
13 ==883==    at 0x4C2DB8F: calloc (in vgpreload_memcheck-amd64-linux.so)
14 ==883==    by 0x402011: mon_printf (mon_programme.c:58)
15 ==883==    by 0x401A4B: prog_hello (mon_programme.c:42)
16 ==883==    by 0x401010: main (mon_programme.c:14)
17 ==883==
18 ==883== 66 bytes in 6 blocks are definitely lost in loss record 5 of 15
19 ==883==    at 0x4C2DB8F: malloc (in vgpreload_memcheck-amd64-linux.so)
20 ==883==    by 0x401B9E: compare_nom (mon_programme.c:65)
21 ==883==    by 0x401042: main (mon_programme.c:16)
22 ==883==
23 ==883==
24 ==883== LEAK SUMMARY:
25 ==883==    definitely lost: 106 bytes in 8 blocks
26 ==883==    indirectly lost: 106 bytes in 8 blocks
27 ==883==      possibly lost: 0 bytes in 0 blocks
28 ==883==    still reachable: 12,384 bytes in 6 blocks
29 ==883==         suppressed: 0 bytes in 0 blocks
30 ==883== Reachable blocks (those to which a pointer was found) are not shown.
31 ==883== To see them, rerun with: --leak-check=full --show-leak-kinds=all
32 ==883==
33 ==883== For counts of detected and suppressed errors, rerun with: -v
34 ==883== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Que pouvez-vous déduire sur le programme `mon_programme` en vous fiant à la sortie Valgrind ?

A Le programme provoque une erreur de segmentation

B Le programme avec l'argument ne provoque pas de fuite mémoire.

■ Le programme avec l'argument `foo` provoque au moins une fuite mémoire.

D Le programme ne provoque pas de fuite mémoire.

E Le programme avec l'argument `foo` provoque une erreur de segmentation

Question 29 ♣ (2 points) Parmi les fonctions suivantes, lesquelles semblent directement à l'origine des fuites ?

☐ A `mon_programme`	☐ C `valgrind`	☐ E `exit`	■ `mon_printf`
☐ B aucune	☐ D `prog_hello`	☐ F `main`	■ `compare_nom`

Question 30 ♣ (2 points) Parmi les fonctions suivantes, lesquelles ont été utilisées pour allouer de la mémoire amenant à une fuite ?

■ `calloc`	☐ B `realloc`	☐ C `alloc`	■ `malloc`

Programmation par contrats, programmation défensive (30 points)

Considérons l'interface **dictionnaire.h** suivante :

```c
#ifndef DICTIONNAIRE_H
#define DICTIONNAIRE_H

#include <stdbool.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);
```

```
78
79  #endif
```

Une implémentation possible est donnée par le code suivant :

```
1   #include "dictionnaire.h"
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5
6   #define LNG_MAX_MOT 30
7
8   // ------------------ //
9   // Fonctions privees //
10  // ------------------ //
11
12  /**
13   * Affiche recursivement le contenu d'un noeud.
14   *
15   * @param noeud       Le noeud a partir duquel on fait l'affichage
16   * @param prefixe     Le mot correpondant au noeud courant
17   * @param profondeur  La profondeur du noeud courant
18   */
19  void Dictionnaire_afficherRecursif(const struct Noeud *noeud,
20                                     char prefixe[LNG_MAX_MOT],
21                                     unsigned int profondeur) {
22      if (noeud != NULL) {
23          if (noeud->terminal) printf("\"%s\" ", prefixe);
24          for (unsigned int i = 0; i < NB_LETTRES; ++i) {
25              prefixe[profondeur] = 'a' + i;
26              prefixe[profondeur + 1] = '\0';
27              Dictionnaire_afficherRecursif(noeud->enfants[i],
28                                            prefixe, profondeur + 1);
29          }
30      }
31  }
32
33  /**
34   * Insere recursivement un mot dans un dictionnaire.
35   *
36   * @param noeud    Le noeud courant
37   * @param suffixe  La portion de mot pas encore lue
38   */
39  void Dictionnaire_insererMotRecursif(struct Noeud **noeud,
40                                       const char *suffixe) {
41      if (*noeud == NULL) {
42          *noeud = malloc(sizeof(struct Noeud));
43          (*noeud)->terminal = false;
44          for (unsigned int i = 0; i < NB_LETTRES; ++i) {
45              (*noeud)->enfants[i] = NULL;
46          }
47      }
48      if (strcmp(suffixe, "") == 0) {
49          (*noeud)->terminal = true;
50      } else {
51          unsigned int i = suffixe[0] - 'a';
52          Dictionnaire_insererMotRecursif(&(*noeud)->enfants[i], suffixe + 1);
53      }
54  }
55
56  /**
57   * Verifie recursivement la presence d'un mot dans un dictionnaire.
58   *
59   * @param noeud    Le noeud courant
60   * @param suffixe  La portion de mot pas encore lue
61   */
62  bool Dictionnaire_contientMotRecursif(const struct Noeud *noeud,
63                                        const char *suffixe) {
64      if (noeud == NULL) {
65          return false;
66      } else if (strcmp(suffixe, "") == 0) {
67          return noeud->terminal;
68      } else {
69          unsigned int i = suffixe[0] - 'a';
70          return Dictionnaire_contientMotRecursif(noeud->enfants[i],
71                                                  suffixe + 1);
72      }
73  }
74
75  /**
76   * Supprime recursivement un mot d'un dictionnaire.
77   *
```

```
78     * @param noeud     Le noeud courant
79     * @param suffixe  La portion de mot pas encore lue
80     */
81    void Dictionnaire_supprimerMotRecursif(struct Noeud *noeud,
82                                           const char *suffixe) {
83        if (noeud != NULL) {
84            if (strcmp(suffixe, "") == 0) {
85                noeud->terminal = false;
86            } else {
87                unsigned int i = suffixe[0] - 'a';
88                Dictionnaire_supprimerMotRecursif(noeud->enfants[i],
89                                                  suffixe + 1);
90            }
91        }
92    }
93
94    /**
95     * Detruit recursivement un dictionnaire.
96     *
97     * @param noeud  Le noeud courant
98     */
99    void Dictionnaire_detruireRecursif(struct Noeud *noeud) {
100       if (noeud != NULL) {
101           for (unsigned int i = 0; i < NB_LETTRES; ++i) {
102               Dictionnaire_detruireRecursif(noeud->enfants[i]);
103           }
104           free(noeud);
105       }
106   }
107
108   // ------------------------------------- //
109   // Implementation des fonctions publiques //
110   // ------------------------------------- //
111
112   struct Dictionnaire Dictionnaire_creer() {
113       struct Dictionnaire dictionnaire = {.racine = NULL,
114                                           .nbMots = 0};
115       return dictionnaire;
116   }
117
118   void Dictionnaire_afficher(const struct Dictionnaire *dictionnaire) {
119       printf("Dictionnaire contenant les mots ");
120       char prefixe[LNG_MAX_MOT + 1] = "";
121       Dictionnaire_afficherRecursif(dictionnaire->racine, prefixe, 0);
122       printf("\n");
123   }
124
125   void Dictionnaire_insererMot(struct Dictionnaire *dictionnaire, const char *mot) {
126       Dictionnaire_insererMotRecursif(&dictionnaire->racine, mot);
127   }
128
129   bool Dictionnaire_contientMot(const struct Dictionnaire *dictionnaire,
130                                 const char *mot) {
131       return Dictionnaire_contientMotRecursif(dictionnaire->racine, mot);
132   }
133
134   void Dictionnaire_supprimerMot(struct Dictionnaire *dictionnaire, const char *mot) {
135       Dictionnaire_supprimerMotRecursif(dictionnaire->racine, mot);
136   }
137
138   void Dictionnaire_detruire(struct Dictionnaire *dictionnaire) {
139       Dictionnaire_detruireRecursif(dictionnaire->racine);
140   }
```

Finalement, considérons le petit programme C suivant qui utilise le module **dictionnaire** :

```
1   #include "dictionnaire.h"
2
3   int main() {
4       struct Dictionnaire dictionnaire;
5       dictionnaire = Dictionnaire_creer();
6       Dictionnaire_insererMot(&dictionnaire, "frais");
7       Dictionnaire_insererMot(&dictionnaire, "banane");
8       Dictionnaire_insererMot(&dictionnaire, "froment");
9       Dictionnaire_insererMot(&dictionnaire, "bandit");
10      Dictionnaire_insererMot(&dictionnaire, "bar");
11      Dictionnaire_insererMot(&dictionnaire, "fromage");
12      Dictionnaire_insererMot(&dictionnaire, "ananas");
13      Dictionnaire_insererMot(&dictionnaire, "barrer");
14      Dictionnaire_insererMot(&dictionnaire, "fraise");
15      Dictionnaire_insererMot(&dictionnaire, "bar");
16      Dictionnaire_insererMot(&dictionnaire, "ange");
```

```
17        Dictionnaire_insererMot(&dictionnaire, "froment");
18        Dictionnaire_supprimerMot(&dictionnaire, "ananas");
19        Dictionnaire_afficher(&dictionnaire);
20        printf("\"ananas\" dans dictionnaire? %s\n",
21               Dictionnaire_contientMot(&dictionnaire, "ananas") ?
22               "oui" : "non");
23        printf("\"tomate\" dans dictionnaire? %s\n",
24               Dictionnaire_contientMot(&dictionnaire, "tomate") ?
25               "oui" : "non");
26        printf("\"anana\" dans dictionnaire? %s\n",
27               Dictionnaire_contientMot(&dictionnaire, "anana") ?
28               "oui" : "non");
29        Dictionnaire_detruire(&dictionnaire);
30        return 0;
31    }
```

Lorsqu'on l'exécute, on obtient le résultat suivant sur la sortie standard :

```
1   Dictionnaire contenant les mots "ange" "banane" "bandit" "bar" "barrer"
2   "frais" "fraise" "fromage" "froment"
3   "ananas" dans dictionnaire? non
4   "tomate" dans dictionnaire? non
5   "anana" dans dictionnaire? non
```

Répondez aux cinq questions suivantes sur la dernière feuille.

Question 31 (2 points) À quoi servent les instructions **#ifndef** DICTIONNAIRE_H, **#define** DICTIONNAIRE_H et **#endif** présentes en début et fin du fichier `dictionnaire.h` ?

Question 32 (3 points) Donnez un exemple d'appel à la fonction Dictionnaire_contientMot qui déclenche une erreur de segmentation à la ligne 131 du fichier `dictionnaire.c`.

Question 33 (5 points) Donnez un exemple d'appel à la fonction Dictionnaire_contientMot qui déclenche une erreur de segmentation (ou, à tout le moins, un comportement imprévisible du programme) à la ligne 70 du fichier `dictionnaire.c`.

Question 34 (10 points) Proposez une correction basée sur la **programmation par contrats** qui corrige le bogue soulevé à la question 32.

Question 35 (10 points) Proposez une correction basée sur la **programmation défensive** qui corrige le bogue soulevé à la question 33.

Réponse (Question 31)

Réponse (Question 32)

Réponse (Question 33)

Réponse (Question 34)

Réponse (Question 35)

Correction

INF3135 **Examen Final - Automne 2017**

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

Question 1 : A B C D ■

Question 2 : A B C D E ■

Question 3 : A B C D ■ F

Question 4 : A B C D ■ F

Question 5 : A B C D ■ F

Question 6 : ■ B C D E F

Question 7 : A ■ C D E F

Question 8 : A B C ■ E F

Question 9 : A B ■ D E F

Question 10 : A ■ C D E F

Question 11 : ■ B C D

Question 12 : ■ B C D E F

Question 13 : A B C ■ E F

Question 14 : A B C ■

Question 15 : A B C D ■ F

Question 16 : A B C D E ■

Question 17 : A B C D E F ■ H I J

Question 18 : A B C D E F G ■ I J K L

Question 19 : A B C ■ E F G H I J K L

Question 20 : ■ B C D E F G H I

Question 21 : A B C D E F G H ■ J K L

Question 22 : A B C D E F G ■ I J K L

Question 23 : A B C D E ■

Question 24 : A B C ■ E F

Question 25 : A B C ■ E F G H

Question 26 : A B C D E ■

Question 27 : ■ B C D E F

Question 28 : A B ■ D E

Question 29 : A B C D E F ■ ■

Question 30 : ■ B C ■

Question 31 :

Question 32 :

Question 33 :

Question 34 :

Question 35 :