



QCM

Langage C

QCM Langage C  
Semestre 6  
Examen Session 1 du  
05/05/2020

Nom et prénom :

SADURNI Thomas

*Durée : 30 minutes.**Les questions faisant apparaître le symbole ♣ peuvent présenter zéro, une ou plusieurs bonnes réponses. Les autres ont une unique bonne réponse.**Une question simple rapporte au maximum 1 point, une question multiple au maximum 3 points.**Des points négatifs pourront être affectés à de très mauvaises réponses.**Pour valider un choix, il faut **cocher** la case.***Seules les cases sont analysées,***il vous est donc possible d'écrire ailleurs sans incidence sur votre rendu.*

## 1 Allocation dynamique

**Question 1** Soient les instructions suivantes :

```
char *initiale = malloc(sizeof(char));  
*initiale = 'A';
```

Cocher l'instruction permettant de libérer la mémoire :

- |   |   |
|---|---|
| <input checked="" type="checkbox"/> <code>free(initiale); initiale = NULL;</code> | <input type="checkbox"/> <code>free(initiale, sizeof(char));</code> |
| <input type="checkbox"/> <code>initiale = NULL;</code>                            | <input type="checkbox"/> <code>initiale.free();</code>              |

**Question 2** On veut pouvoir enregistrer 6 réels de plus dans un tableau de T réels, tableau alloué dynamiquement. Un étudiant propose cette instruction qui compile et s'exécute sans erreur :

```
float *tab = realloc(tab, (T+6)*sizeof(float));
```

Pourquoi cet étudiant se trompe-t-il ?

- ☐ Il faut indiquer uniquement `6 * sizeof(float)` en second paramètre de l'appel à `realloc`.
- ☒ En cas d'échec de la réallocation, `realloc` retourne `NULL` et on aura perdu l'adresse de la mémoire initiale dans `tab`.

**Question 3** Que signifie le fait que l'allocateur `malloc` retourne l'adresse `NULL` ?

- ☒ L'allocateur n'a pas réussi à allouer la mémoire demandée de façon contigüe.
- ☐ L'allocateur vous indique qu'il faut allouer de la mémoire dans la pile.



**Question 4 ♣** L'allocateur `realloc` permet de modifier la taille mémoire allouée dynamiquement à une adresse donnée. Voici sa signature :

```
void* realloc(void* ptr_mem, size_t taille)
```

Cocher la ou les propositions justes :

- ☒ `realloc` retourne `NULL` ou l'adresse d'une zone mémoire de `taille` octets.
- ☒ Si `realloc` retourne l'adresse `NULL`, alors la reallocation a échoué. La zone mémoire à l'adresse `ptr_mem` reste allouée.
- ☐ `taille` représente l'incrément de taille mémoire demandé.
- ☐ `ptr_mem` contient l'adresse de la zone mémoire après réallocation (mode in out).
- ☐ Aucune de ces réponses n'est correcte.

**Question 5** Voici la définition de la procédure `malloc` :

```
void* malloc(size_t taille);
```

Quelle proposition caractérise le type `size_t` du paramètre `taille` ?

- ☒ c'est un alias de `unsigned int`
- ☐ la taille mémoire demandée en octet
- ☐ la taille mémoire demandée en bit

**Question 6 ♣** Pour le jeu d'instructions suivant :

```
enum outil {BECHE, PELLE, SEAU, ARROSOIR};  
enum outil *ustensile;  
ustensile = calloc(1, sizeof(enum outil));  
assert(*ustensile == XXX);
```

Cocher une valeur pour `XXX` qui valide l'assert.

- ☐ `NULL`
- ☒ `0`
- ☐ `BECHE`
- ☐ Aucune de ces réponses n'est correcte.

**Question 7 ♣** Cocher la ou les instructions correctes qui permettent d'allouer de l'espace pour enregistrer un réel avec `malloc`.

- ☐ `float *v = malloc( sizeof(*v));`
- ☐ `float *v = malloc( sizeof(float));`
- ☒ `float v = malloc( sizeof(float));`
- ☐ `float *v = malloc( float );`
- ☐ Aucune de ces réponses n'est correcte.

**Question 8** Voici la définition de la procédure `malloc` :

```
void* malloc(size_t taille);
```

Le type de retour est `void*`. Quelle en est la conséquence ?

- ☐ on peut affecter tout type de pointeur avec le retour de `malloc`.
- ☒ on ne peut allouer que des zones mémoires vides.



**Question 9 ♣** Soient les instructions suivantes :

```
char *initiale = malloc(sizeof(char));
*initiale = 'A';
free(initiale);
printf("L'initiale est %c", *initiale);
```

Qu'affiche la dernière instruction `printf` ?

- ☐ 'A' ☐ Probablement 'A'
- ☐ Ce code est faux. ☒ Aucune de ces réponses n'est correcte.
- ☐ L'exécution échoue à cause d'une erreur de segmentation

**Question 10 ♣** On souhaite allouer \*\*\* dynamiquement \*\*\* une variable tableau de 15 caractères. Cocher la ou les bonne(s) instruction(s) :

- ☒ `char *t = calloc(15, sizeof(char));`
- ☒ `char *t = malloc(15 * sizeof(char));`
- ☐ `char *t = 15 * malloc(sizeof(char));`
- ☒ `char *t = malloc(15 * sizeof(*t));`
- ☐ Aucune de ces réponses n'est correcte.

**Question 11 ♣** Voici la définition de la procédure `calloc` :

```
void* calloc(size_t nb, size_t taille_element);
```

Cocher la ou les utilisations convenables de `calloc` pour allouer dynamiquement un réel :

- ☒ `float *y = calloc(1, sizeof(float));`
- ☒ `float *y = calloc(1, sizeof(*y) );`
- ☐ `float *y = calloc( sizeof(float), 1);`
- ☐ `float *y = calloc( 1 * sizeof(float));`
- ☐ Aucune de ces réponses n'est correcte.

## 2 Les modules

**Question 12** L'interface du module `date.h` présente la structure suivante :

```
#ifndef DATE__H
#define DATE__H
struct Date {
    int jour;
    int mois;
};
# endif
```

Quel est le rôle des commandes pré-processeur `#ifndef`, `#define` et `#endif` ?

- ☐ De définir un type `struct Date` que le pré-processeur peut exploiter.
- ☒ De pouvoir compiler même si `date.h` est inclus plusieurs fois.



**Question 13 ♣** Pour pouvoir générer un exécutable à partir de plusieurs modules et d'un programme principal, le compilateur vérifie un ensemble de contraintes. Parmi les contraintes suivantes, cocher les contraintes qui **empêchent** la production de l'exécutable :

- ☒ Absence du sous-programme `int main()` dans les fichiers `.c`
- ☒ Utilisation multiple d'un même identificateur pour définir un sous-programme, une variable ou un type.
- ☐ La déclaration d'une variable globale dans un module.
- ☐ Aucune de ces réponses n'est correcte.

**Question 14** On souhaite définir un module `pile` en C. Quels fichiers doit-on créer par convention ?

- ☒ Pour l'interface `pile.h` et `pile.c` pour le corps
- ☐ Pour l'interface `pile.c` et `pile.h` pour le corps
- ☐ Pour l'interface `pile.h` et `pile.cc` pour le corps

**Question 15** On souhaite utiliser le module `date` en C dans le fichier `principal.c`. Quelle instruction doit-on ajouter au début de `principal.c` ?

- ☐ `#include <date.h>`
- ☒ `#include "date.h"`
- ☐ `include "date.h"`

**Question 16** Le corps du module `date.c` présente la fonction suivante :

```
static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

Le programme principal `visualiser.c` inclut `date.h`. Peut-il utiliser le sous-programme `max` dans `visualiser.c` ?

- ☐ Oui, mais seulement si la garde conditionnelle est présente dans `date.h`.
- ☒ Non
- ☐ Oui

**Question 17** Est-ce que la commande suivante produit un exécutable ? On suppose qu'il n'y a pas d'erreur dans les programmes.

`c99 -Wextra -pedantic afficher.c fraction.c -o`

- ☐ Oui, mais il aura un nom par défaut.
- ☒ Non.



### 3 Make

**Question 18** Les premières règles d'un fichier **Makefile** sont les suivantes :

```
all: test_file exemple_file

test_file: test_file.o file.o
    c99 test_file.o file.o -o test_file

exemple_file: exemple_file.o file.o
    c99 exemple_file.o file.o -o exemple_file
```

Quelle est la première **commande exécutée** par la commande **make** [on supposera qu'on lance **make** pour la première fois]?

- ☐ c99 exemple\_file.o file.o -o exemple\_file  
☒ c99 test\_file.o file.o -o test\_file

**Question 19 ♣** Soit la règle suivante :

```
a:b c
xxx
```

La commande **xxx** sera exécutée :

- ☐ si c est plus récent que a  
☐ si a n'existe pas [et b et c existent]
- ☒ si a est plus récent que c  
☐ Aucune de ces réponses n'est correcte.

**Question 20** Voici une des règles explicites listées dans un **makefile** :

```
date.o: date.c date.h
    c99 -Wextra -pedantic -c date.c
```

Quelles sont les dépendances de cette règle ?

- ☒ date.c date.h  
☐ date.o

**Question 21** Dans la règle suivante, que désigne **\$@** ?

```
main: main.c
    ${CC} ${CFLAGS} ${LDFLAGS} $< -o $@
```

- ☐ un nouveau nom  
☒ main
- ☐ main.c