

QCM

Langage C

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

Nom et prénom :

.....

*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 :

```
int *valeur = malloc(sizeof(int));
*valeur = 10;
free(valeur);
printf("la valeur est %d", *valeur);
```

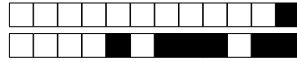
Qu'affiche la dernière instruction `printf` ?☐ 10☐ Ce code est faux.☐ Probablement 10☐ L'exécution échoue à cause d'une
erreur de segmentation☐ Aucune de ces réponses n'est correcte.**Question 2** Voici la définition de la procédure `malloc` :

```
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 3 ♣** 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 4 ♣ 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(*y));`
- ☐ `float *y = calloc(sizeof(float), 1);`
- ☐ `float *y = calloc(1 * sizeof(float));`
- ☐ `float *y = calloc(1, sizeof(float));`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 5 On veut pouvoir enregistrer 10 entiers de plus dans un tableau de `T` entiers, tableau alloué dynamiquement. Un étudiant propose cette instruction qui compile et s'exécute sans erreur :

```
int *tab = realloc(tab, (T+10)*sizeof(int));
```

Pourquoi cet étudiant se trompe-t-il ?

- ☐ En cas d'échec de la réallocation, `realloc` retourne `NULL` et on aura perdu l'adresse de la mémoire initiale dans `tab`.
- ☐ Il faut indiquer uniquement `10 * sizeof(int)` en second paramètre de l'appel à `realloc`.

Question 6 ♣ Cocher la ou les instructions correctes qui permettent d'allouer de l'espace pour enregistrer un caractère avec `malloc`.

- ☐ `char *ch = malloc(sizeof(char));`
- ☐ `char *ch = malloc(char);`
- ☐ `char ch = malloc(sizeof(char));`
- ☐ `char *ch = malloc(sizeof(*ch));`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 7 ♣ 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.
- ☐ `ptr_mem` contient l'adresse de la zone mémoire après réallocation (mode in out).
- ☐ `taille` représente l'incrément de taille mémoire demandé.
- ☐ Si `realloc` retourne l'adresse `NULL`, alors la réallocation a échoué. La zone mémoire à l'adresse `ptr_mem` reste allouée.
- ☐ *Aucune de ces réponses n'est correcte.*

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

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

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

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



Question 9 Comment savoir si l'allocation dynamique a réussi ?

- ☐ On vérifie si l'allocateur ne retourne pas le pointeur NULL.
- ☐ L'exécution se déroule sans erreur.

Question 10 Soient les instructions suivantes :

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

Cocher l'instruction permettant de libérer la mémoire :

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

Question 11 ♣ On souhaite allouer *** dynamiquement *** une variable tableau de 15 caractères. Cocher la ou les bonne(s) instruction(s) :

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

2 Les modules

Question 12 Le corps du module `date.c` présente la fonction suivante :

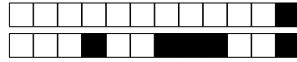
```
static int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

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

- | | |
|---|------------------------------|
| <input type="checkbox"/> Oui, mais seulement si la garde conditionnelle est présente dans <code>date.h</code> . | <input type="checkbox"/> Non |
| | <input type="checkbox"/> Oui |

Question 13 On souhaite utiliser le module `pile` en C dans le fichier `principal.c`. Quelle instruction doit-on ajouter au début de `principal.c` ?

- ☐ `#include <pile.h>`
- ☐ `#import "pile.h"`
- ☐ `#include "pile.h"`



Question 14 ♣ 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 :

- ☐ La déclaration d'une constante pré-processeur dans le corps d'un module.
- ☐ La déclaration en-avant d'un sous-programme dans un module.
- ☐ Utilisation d'un même identificateur pour définir plusieurs sous-programmes différents.
- ☐ Aucune de ces réponses n'est correcte.

Question 15 On souhaite définir un module `pile` en C. Quels fichiers doit-on créer par convention ?

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

Question 16 Dans quelle partie du module date trouve-t-on typiquement le type d'instructions suivantes :

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

- ☐ Dans le corps `date.c`.
- ☐ Dans l'interface `date.h`.

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 -c liste.c
```

- ☐ Oui, si un sous-programme `int main()` existe dans `liste.c`. ☐ Non

3 Make

Question 18 Une règle dans un `makefile` suit la structure suivante :

```
a:b
c
```

Quels sont les termes qui décrivent respectivement les parties `a`, `b` et `c` d'une règle ?

- ☐ cible, dépendance, commande
- ☐ commande, cible, dépendance
- ☐ dépendance, cible, commande



Question 19 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 20 ♣ Soit la règle suivante :

```
a:b c
xxx
```

La commande **xxx** sera exécutée :

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

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

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

- ☐ main.c ☐ main
☐ un nouveau nom