Sujet n^o 1

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. `printf("%i", A)`
   a. Affiche le caractère dont le code ASCII est stocké dans la variable A
   b. Affiche le code ASCII du caractère stocké dans la variable A
   c. Affiche le code ASCII du caractère `'A'`
   d. Affiche le caractère `'A'`

2. `printf("%c", 'A')`
   a. Affiche le code ASCII du caractère `'A'`
   b. Affiche le caractère dont le code ASCII est stocké dans la variable A
   c. Affiche le caractère `'A'`
   d. Affiche le code ASCII du caractère stocké dans la variable A

3. L'instruction
```
for(i=1 ; i<=n ; i++)
    printf("%i", n-i) ;
```
   a. affiche les entiers de 0 à n-1 en ordre croissant
   b. affiche les entiers de 0 à n-1 en ordre décroissant
   c. affiche les entiers de 1 à n en ordre croissant
   d. affiche les entiers de 1 à n en ordre décroissant

4. En langage C, une fonction qui ne renvoie rien, c'est :
   a. une fonction de type `void`
   b. une fonction de type pointeur
   c. une fonction de type `int`
   d. impossible

5. Si on ajoute 1 au nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 101011010101000
   b. 101011010101111
   c. 101011010100110
   d. 101011010100000

6. On définit une fonction par
```
int f(int x){
    return 3*x+1 ;
}
```
Dans main(), l'expression f(n)
   a. est logiquement équivalente à 3*n+1
   b. a pour valeur 3*n+1
   c. est égale à VRAI ou FAUX
   d. affiche 3*n+1

7. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un mot-clef du langage C
   c. Un opérateur du langage C
   d. Un identificateur du langage C

8. `if` est :
   a. Un identificateur du langage C
   b. Un mot-clef du langage C
   c. Une commande qu'on tape dans la fenêtre de commande
   d. Un opérateur du langage C

9. Après `char c ; c='a' ; c=c+1 ;`
   a. Un message d'erreur s'affiche
   b. c vaut `'a'`
   c. c vaut `'b'`
   d. c vaut `'A'`

10. L'instruction
```
for(i=1 ; i<10 ; i++){
    printf("%i ", i) ;
    printf("%i ", i*i) ;
}
```
   a. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   b. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré
   c. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré
   d. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100

11. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
   a. Il y a une faute d'orthographe
   b. On utilise `%i` car a est de type `int`
   c. Il y a une faute de syntaxe
   d. `%` est un opérateur permettant de calculer un reste

12. L'expression `a<=1`
   a. Diminue de 1 la valeur de a
   b. Utilise les opérateurs `<` et `=`
   c. A pour valeur VRAI si $a \leq 1$ et FAUX sinon
   d. Réalise une affectation

13. `if (a<5) printf("Bonjour") ; a=a+1 ;`
   a. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
   b. Affiche bonjour et augmente la valeur de a quelque soit a
   c. Augmente la valeur de a quelque soit a
   d. Affiche bonjour quelque soit a

14. `=` permet en langage C de :
   a. tester une égalité
   b. comparer deux identificateurs
   c. réaliser une affectation
   d. convertir un int en float

15. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
   a. à l'exécution, 0 s'affichera
   b. à l'exécution, 1 s'affichera
   c. à l'exécution, n'importe quel nombre s'affichera
   d. à la compilation, un message d'erreur s'affichera

16. `if (a%2 == 0) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier impair
    b. Affiche bonjour quand a est un entier pair
    c. N'affiche rien (quelque soit la valeur de a)
    d. Déclenche le message d'erreur `invalid lvalue in assignment`

17. En typographie le caractère `&` s'appelle :
    a. L'espagnolette
    b. L'arpette
    c. L'esperluette
    d. La luette

18. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
    a. après la dernière ligne }
    b. à la fin de la ligne `printf("%i", tab[i]) ;`
    c. il est impossible de rajouter } tout en gardant le même comportement
    d. à la fin de la ligne `printf("\ n") ;`

19. En langage C, #include<stdio.h> permet
    a. d'accélérer l'exécution
    b. d'accélérer la compilation
    c. est obligatoire dans tout programme de langage C
    d. d'utiliser des fonctions d'entrée-sortie

20. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```
    a. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré
    b. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
    c. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré
    d. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100

21. `if (a%2 = 0) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier pair
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. Affiche bonjour quand a est un entier impair
    d. N'affiche rien (quelque soit la valeur de a)

22. La ligne de programme `scanf("%i", &a) ;`
    a. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
    b. Contient une erreur
    c. Affiche l'adresse mémoire de la variable `a`
    d. Permet de stocker dans `a` un entier au choix de l'utilisateur

23. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 1010110101001111
    b. 1010110101010000
    c. 1010110101001110
    d. 1010110101011110

24. Dans le programme de l'exercice 1, `MAX`
    a. est un pointeur
    b. est de type `int`
    c. est remplacé par 10 partout où il apparaît lors de la compilation
    d. est de type `float`

25. Le nombre qui se note 110101 en base 2 se note en base 10 :
    a. 35
    b. 63
    c. 36
    d. 53

26. En langage C, `++` est
    a. Un opérateur d'incrémentation
    b. Un opérateur de décrémentation
    c. Un opérateur d'accrétion
    d. Un opérateur de décrétinisation

27. L'expression `(0 == 7%3) && (0 == 9%3)`
    a. a pour valeur VRAI
    b. entraîne l'affichage d'un message d'erreur
    c. a pour valeur FAUX
    d. n'a pas de valeur

28. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
    a. $i!$ à la case n$^{\text{o}}$ i
    b. $2i$ à la case n$^{\text{o}}$ i
    c. 2^i à la case n$^{\text{o}}$ i
    d. le i^{e} nombre de Fibonacci à la case n$^{\text{o}}$ i

29. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
    a. 111
    b. 100
    c. 101
    d. 110

30. `printf("%i", 'A')`
    a. Affiche le code ASCII du caractère `'A'`
    b. Affiche le caractère dont le code ASCII est stocké dans la variable A
    c. Affiche le code ASCII du caractère stocké dans la variable A
    d. Affiche le caractère `'A'`

31. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
    a. Utilisent un opérateur d'incrémentation
    b. Sont bien indentées
    c. Demandent un entier à l'utilisateur si `i` vaut 1
    d. Contiennent des erreurs

32. `gcc` est :
    a. Un mot-clef du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un identificateur du langage C
    d. Un opérateur du langage C

33. `printf("%c", A)`
    a. Affiche le code ASCII du caractère stocké dans la variable A
    b. Affiche le code ASCII du caractère `'A'`
    c. Affiche le caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

34. Après `int t[10] ;`
    a. `t` est un tableau de dix cases numérotées de 1 à 10
    b. `t` est un entier compris entre 0 et 9
    c. `t` est un entier compris entre 1 et 10
    d. `t` est un tableau de dix cases numérotées de 0 à 9

35. La ligne de programme `printf("%i, %f, %c", a, b, c) ;`
    a. Affiche les nombres `a, b, c` sous trois formats différents
    b. Contient une erreur
    c. Affiche les caractères `'i', 'f', 'c'`
    d. Affiche l'entier `a`, le réel `b` et le caractère `c`

36. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
    a. $\lfloor \log_2(n) \rfloor$
    b. $\lfloor \log_2(n) \rfloor - 1$
    c. $\lfloor \log_2(n) \rfloor + 1$
    d. $\lfloor \log_2(n) - 1 \rfloor$

37. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
    a. `a` vaut 5 et `b` a une valeur indéterminée
    b. `a` a une valeur indéterminée et `b` vaut 5
    c. `a` vaut 5 et `b` vaut 5
    d. `a` vaut 5 et `b` a une valeur indéterminée

38. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
    a. Un opérateur du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un identificateur du langage C
    d. Un mot-clef du langage C

39. Après `int a, b ; a=3/2 ; b=3%2 ;`
    a. `a` vaut 1.5 et `b` vaut 0
    b. `a` vaut 1 et `b` vaut 1
    c. `a` vaut 1 et `b` vaut 0
    d. `a` vaut 1.5 et `b` vaut 1

40. `if (a%2) printf("bonjour") ;`
    a. Déclenche le message d'erreur `invalid lvalue in assignment`
    b. N'affiche rien (quelque soit le type et la valeur de a)
    c. Affiche bonjour quand a est un entier impair
    d. Affiche bonjour quand a est un entier pair

Sujet n^o 2

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. `printf("%c", 'A')`
   a. Affiche le caractère `'A'`
   b. Affiche le code ASCII du caractère stocké dans la variable A
   c. Affiche le caractère dont le code ASCII est stocké dans la variable A
   d. Affiche le code ASCII du caractère `'A'`

2. Après `int a, b ; a=3/2 ; b=3%2 ;`
   a. `a` vaut 1.5 et `b` vaut 0
   b. `a` vaut 1.5 et `b` vaut 1
   c. `a` vaut 1 et `b` vaut 1
   d. `a` vaut 1 et `b` vaut 0

3. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
   a. Demandent un entier à l'utilisateur si `i` vaut 1
   b. Contiennent des erreurs
   c. Sont bien indentées
   d. Utilisent un opérateur d'incrémentation

4. L'expression `a<=1`
   a. Réalise une affectation
   b. Utilise les opérateurs `<` et `=`
   c. A pour valeur VRAI si $a \leq 1$ et FAUX sinon
   d. Diminue de 1 la valeur de a

5. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
   a. 111
   b. 110
   c. 101
   d. 100

6. En typographie le caractère `&` s'appelle :
   a. L'arpette
   b. L'espagnolette
   c. L'esperluette
   d. La luette

7. On définit une fonction par
```
int f(int x){
  return 3*x+1 ;
}
```
Dans main(), l'expression f(n)
   a. a pour valeur 3*n+1
   b. est logiquement équivalente à 3*n+1
   c. affiche 3*n+1
   d. est égale à VRAI ou FAUX

8. L'instruction `for(i=1 ; i<=n ; i++)`
`printf("%i", n-i) ;`
   a. affiche les entiers de 0 à n-1 en ordre décroissant
   b. affiche les entiers de 1 à n en ordre décroissant
   c. affiche les entiers de 1 à n en ordre croissant
   d. affiche les entiers de 0 à n-1 en ordre croissant

9. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
   a. Un identificateur du langage C
   b. Un opérateur du langage C
   c. Une commande qu'on tape dans la fenêtre de commande
   d. Un mot-clef du langage C

10. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
   a. à la fin de la ligne `printf("%i", tab[i]) ;`
   b. à la fin de la ligne `printf("\ n") ;`
   c. après la dernière ligne }
   d. il est impossible de rajouter } tout en gardant le même comportement

11. `printf("%i", 'A')`
   a. Affiche le caractère `'A'`
   b. Affiche le caractère dont le code ASCII est stocké dans la variable A
   c. Affiche le code ASCII du caractère `'A'`
   d. Affiche le code ASCII du caractère stocké dans la variable A

12. L'instruction
```
for(i=1 ; i<10 ; i++){
  printf("%i ", i) ;
  printf("%i ", i*i) ;
}
```
   a. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré
   b. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   c. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   d. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré

13. Si on ajoute 1 au nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 101011010100000
   b. 101011010101111
   c. 101011010100110
   d. 101011010101000

14. En langage C, `++` est
   a. Un opérateur d'accrétion
   b. Un opérateur d'incrémentation
   c. Un opérateur de décrétinisation
   d. Un opérateur de décrémentation

15. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
   a. `a` vaut 5 et `b` vaut 5
   b. `a` vaut 5 et `b` a une valeur indéterminée
   c. `a` vaut 5 et `b` a une valeur indéterminée
   d. `a` a une valeur indéterminée et `b` vaut 5

16. `=` permet en langage C de :
    a. tester une égalité
    b. comparer deux identificateurs
    c. réaliser une affectation
    d. convertir un int en float

17. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
    a. `%` est un opérateur permettant de calculer un reste
    b. On utilise `%i` car a est de type `int`
    c. Il y a une faute de syntaxe
    d. Il y a une faute d'orthographe

18. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
    a. 2^i à la case n^o i
    b. $i!$ à la case n^o i
    c. le i^e nombre de Fibonacci à la case n^o i
    d. $2i$ à la case n^o i

19. Le nombre qui se note 110101 en base 2 se note en base 10 :
    a. 63
    b. 35
    c. 53
    d. 36

20. Après `int t[10] ;`
    a. `t` est un entier compris entre 1 et 10
    b. `t` est un tableau de dix cases numérotées de 0 à 9
    c. `t` est un entier compris entre 0 et 9
    d. `t` est un tableau de dix cases numérotées de 1 à 10

21. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
    a. à la compilation, un message d'erreur s'affichera
    b. à l'exécution, 0 s'affichera
    c. à l'exécution, n'importe quel nombre s'affichera
    d. à l'exécution, 1 s'affichera

22. `if` est :
    a. Un identificateur du langage C
    b. Un mot-clef du langage C
    c. Un opérateur du langage C
    d. Une commande qu'on tape dans la fenêtre de commande

23. `if (a%2 == 0) printf("bonjour") ;`
    a. N'affiche rien (quelque soit la valeur de a)
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. Affiche bonjour quand a est un entier impair
    d. Affiche bonjour quand a est un entier pair

24. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
    a. Une commande qu'on tape dans la fenêtre de commande
    b. Un mot-clef du langage C
    c. Un identificateur du langage C
    d. Un opérateur du langage C

25. L'expression `(0 == 7%3) && (0 == 9%3)`
    a. n'a pas de valeur
    b. entraîne l'affichage d'un message d'erreur
    c. a pour valeur FAUX
    d. a pour valeur VRAI

26. `if (a%2) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier pair
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. N'affiche rien (quelque soit le type et la valeur de a)
    d. Affiche bonjour quand a est un entier impair

27. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```
    a. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
    b. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré
    c. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré
    d. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100

28. Après `char c ; c='a' ; c=c+1 ;`
    a. Un message d'erreur s'affiche
    b. c vaut `'A'`
    c. c vaut `'b'`
    d. c vaut `'a'`

29. `if (a<5) printf("Bonjour") ; a=a+1 ;`
    a. Augmente la valeur de a quelque soit a
    b. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
    c. Affiche bonjour et augmente la valeur de a quelque soit a
    d. Affiche bonjour quelque soit a

30. Dans le programme de l'exercice 1, `MAX`
    a. est de type `int`
    b. est de type `float`
    c. est remplacé par 10 partout où il apparaît lors de la compilation
    d. est un pointeur

31. `if (a%2 = 0) printf("bonjour") ;`
    a. N'affiche rien (quelque soit la valeur de a)
    b. Affiche bonjour quand a est un entier pair
    c. Déclenche le message d'erreur `invalid lvalue in assignment`
    d. Affiche bonjour quand a est un entier impair

32. En langage C, #include<stdio.h> permet
    a. est obligatoire dans tout programme de langage C
    b. d'accélérer la compilation
    c. d'utiliser des fonctions d'entrée-sortie
    d. d'accélérer l'exécution

33. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
    a. $\lfloor \log_2(n) - 1 \rfloor$
    b. $\lfloor \log_2(n) \rfloor + 1$
    c. $\lfloor \log_2(n) \rfloor - 1$
    d. $\lfloor \log_2(n) \rfloor$

34. `printf("%c", A)`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le code ASCII du caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

35. En langage C, une fonction qui ne renvoie rien, c'est :
    a. une fonction de type pointeur
    b. impossible
    c. une fonction de type `void`
    d. une fonction de type `int`

36. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 1010110101001110
    b. 1010110101001111
    c. 1010110101011110
    d. 1010110101010000

37. La ligne de programme `scanf("%i", &a);`
    a. Affiche l'adresse mémoire de la variable `a`
    b. Contient une erreur
    c. Permet de stocker dans `a` un entier au choix de l'utilisateur
    d. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`

38. `printf("%i", A)`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le caractère dont le code ASCII est stocké dans la variable A
    d. Affiche le code ASCII du caractère `'A'`

39. La ligne de programme `printf("%i, %f, %c", a, b, c);`
    a. Affiche les caractères `'i', 'f', 'c'`
    b. Affiche les nombres `a, b, c` sous trois formats différents
    c. Contient une erreur
    d. Affiche l'entier `a`, le réel `b` et le caractère `c`

40. `gcc` est :
    a. Un identificateur du langage C
    b. Un mot-clef du langage C
    c. Une commande qu'on tape dans la fenêtre de commande
    d. Un opérateur du langage C

Sujet n^o 3

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. L'instruction
```
for(i=1 ; i<=n ; i++)
    printf("%i", n-i) ;
```
   a. affiche les entiers de 1 à n en ordre décroissant
   b. affiche les entiers de 0 à n-1 en ordre décroissant
   c. affiche les entiers de 0 à n-1 en ordre croissant
   d. affiche les entiers de 1 à n en ordre croissant

2. En langage C, #include<stdio.h> permet
   a. d'accélérer la compilation
   b. d'accélérer l'exécution
   c. d'utiliser des fonctions d'entrée-sortie
   d. est obligatoire dans tout programme de langage C

3. En langage C, `++` est
   a. Un opérateur d'accrétion
   b. Un opérateur d'incrémentation
   c. Un opérateur de décrémentation
   d. Un opérateur de décrétinisation

4. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
   a. le i^{e} nombre de Fibonacci à la case n$^{\text{o}}$ i
   b. $i!$ à la case n$^{\text{o}}$ i
   c. $2i$ à la case n$^{\text{o}}$ i
   d. 2^i à la case n$^{\text{o}}$ i

5. La ligne de programme `scanf("%i", &a) ;`
   a. Contient une erreur
   b. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
   c. Affiche l'adresse mémoire de la variable `a`
   d. Permet de stocker dans `a` un entier au choix de l'utilisateur

6. L'instruction
```
for(i=1 ; i<10 ; i++){
    printf("%i ", i) ;
    printf("%i ", i*i) ;
}
```
   a. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   b. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré
   c. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   d. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré

7. L'expression `a<=1`
   a. Utilise les opérateurs `<` et `=`
   b. Réalise une affectation
   c. A pour valeur VRAI si $a \leq 1$ et FAUX sinon
   d. Diminue de 1 la valeur de a

8. `gcc` est :
   a. Un mot-clef du langage C
   b. Un opérateur du langage C
   c. Une commande qu'on tape dans la fenêtre de commande
   d. Un identificateur du langage C

9. `if (a%2 == 0) printf("bonjour") ;`
   a. Affiche bonjour quand a est un entier pair
   b. N'affiche rien (quelque soit la valeur de a)
   c. Déclenche le message d'erreur `invalid lvalue in assignment`
   d. Affiche bonjour quand a est un entier impair

10. La ligne de programme `printf("%i, %f, %c", a, b, c) ;`
   a. Affiche l'entier `a`, le réel `b` et le caractère `c`
   b. Affiche les caractères `'i', 'f', 'c'`
   c. Contient une erreur
   d. Affiche les nombres `a, b, c` sous trois formats différents

11. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
   a. 111
   b. 101
   c. 110
   d. 100

12. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un opérateur du langage C
   c. Un identificateur du langage C
   d. Un mot-clef du langage C

13. Après `int a, b ; a=3/2 ; b=3%2 ;`
   a. `a` vaut 1 et `b` vaut 1
   b. `a` vaut 1.5 et `b` vaut 0
   c. `a` vaut 1.5 et `b` vaut 1
   d. `a` vaut 1 et `b` vaut 0

14. Le nombre qui se note 110101 en base 2 se note en base 10 :
   a. 53
   b. 63
   c. 36
   d. 35

15. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
   a. $\lfloor \log_2(n) \rfloor + 1$
   b. $\lfloor \log_2(n) \rfloor - 1$
   c. $\lfloor \log_2(n) \rfloor$
   d. $\lfloor \log_2(n) - 1 \rfloor$

16. L'expression `(0 == 7%3) && (0 == 9%3)`
    a. entraîne l'affichage d'un message d'erreur
    b. n'a pas de valeur
    c. a pour valeur FAUX
    d. a pour valeur VRAI

17. `printf("%i", A)`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère `'A'`
    c. Affiche le caractère dont le code ASCII est stocké dans la variable A
    d. Affiche le code ASCII du caractère stocké dans la variable A

18. `printf("%c", A)`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le code ASCII du caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

19. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
    a. Utilisent un opérateur d'incrémentation
    b. Demandent un entier à l'utilisateur si `i` vaut 1
    c. Sont bien indentées
    d. Contiennent des erreurs

20. `=` permet en langage C de :
    a. comparer deux identificateurs
    b. tester une égalité
    c. convertir un int en float
    d. réaliser une affectation

21. `printf("%i", 'A')`
    a. Affiche le caractère dont le code ASCII est stocké dans la variable A
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le caractère `'A'`
    d. Affiche le code ASCII du caractère `'A'`

22. En typographie le caractère `&` s'appelle :
    a. L'esperluette
    b. L'espagnolette
    c. La luette
    d. L'arpette

23. On définit une fonction par
```
int f(int x){
  return 3*x+1 ;
}
```
Dans main(), l'expression f(n)
   a. a pour valeur 3*n+1
   b. est logiquement équivalente à 3*n+1
   c. est égale à VRAI ou FAUX
   d. affiche 3*n+1

24. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
   a. `a` a une valeur indéterminée et `b` vaut 5
   b. `a` vaut 5 et `b` a une valeur indéterminée
   c. `a` vaut 5 et `b` a une valeur indéterminée
   d. `a` vaut 5 et `b` vaut 5

25. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```
   a. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré
   b. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
   c. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
   d. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré

26. `if (a%2 = 0) printf("bonjour") ;`
   a. Affiche bonjour quand a est un entier impair
   b. Affiche bonjour quand a est un entier pair
   c. N'affiche rien (quelque soit la valeur de a)
   d. Déclenche le message d'erreur `invalid lvalue in assignment`

27. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 1010110101001110
   b. 1010110101010000
   c. 1010110101001111
   d. 1010110101011110

28. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
   a. On utilise `%i` car a est de type `int`
   b. Il y a une faute de syntaxe
   c. Il y a une faute d'orthographe
   d. `%` est un opérateur permettant de calculer un reste

29. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
   a. à l'exécution, 0 s'affichera
   b. à l'exécution, n'importe quel nombre s'affichera
   c. à l'exécution, 1 s'affichera
   d. à la compilation, un message d'erreur s'affichera

30. `if (a<5) printf("Bonjour") ; a=a+1 ;`
    a. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
    b. Affiche bonjour quelque soit a
    c. Affiche bonjour et augmente la valeur de a quelque soit a
    d. Augmente la valeur de a quelque soit a

31. `printf("%c", 'A')`
    a. Affiche le code ASCII du caractère stocké dans la variable A
    b. Affiche le caractère dont le code ASCII est stocké dans la variable A
    c. Affiche le code ASCII du caractère `'A'`
    d. Affiche le caractère `'A'`

32. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
    a. Un mot-clef du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un identificateur du langage C
    d. Un opérateur du langage C

33. En langage C, une fonction qui ne renvoie rien, c'est :
    a. impossible
    b. une fonction de type `void`
    c. une fonction de type `int`
    d. une fonction de type pointeur

34. `if (a%2) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier impair
    b. Affiche bonjour quand a est un entier pair
    c. Déclenche le message d'erreur `invalid lvalue in assignment`
    d. N'affiche rien (quelque soit le type et la valeur de a)

35. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
    a. après la dernière ligne }
    b. il est impossible de rajouter } tout en gardant le même comportement
    c. à la fin de la ligne `printf("%i", tab[i]) ;`
    d. à la fin de la ligne `printf("\ n") ;`

36. Dans le programme de l'exercice 1, `MAX`
    a. est de type `float`
    b. est remplacé par 10 partout où il apparaît lors de la compilation
    c. est un pointeur
    d. est de type `int`

37. `if` est :
    a. Un identificateur du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un opérateur du langage C
    d. Un mot-clef du langage C

38. Après `char c ; c='a' ; c=c+1 ;`
    a. c vaut `'A'`
    b. c vaut `'b'`
    c. Un message d'erreur s'affiche
    d. c vaut `'a'`

39. Si on ajoute 1 au nombre qui se note en base 2 1010110101001111, on obtient le nombre qui se note en base 2 :
    a. 1010110101001110
    b. 1010110101010000
    c. 1010110101011111
    d. 1010110101000000

40. Après `int t[10] ;`
    a. `t` est un tableau de dix cases numérotées de 0 à 9
    b. `t` est un entier compris entre 0 et 9
    c. `t` est un tableau de dix cases numérotées de 1 à 10
    d. `t` est un entier compris entre 1 et 10

Sujet n^o 4

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. `printf("%i", A)`
   a. Affiche le code ASCII du caractère `'A'`
   b. Affiche le code ASCII du caractère stocké dans la variable A
   c. Affiche le caractère dont le code ASCII est stocké dans la variable A
   d. Affiche le caractère `'A'`

2. Dans le programme de l'exercice 1, `MAX`
   a. est remplacé par 10 partout où il apparaît lors de la compilation
   b. est de type `int`
   c. est de type `float`
   d. est un pointeur

3. Après `char c ; c='a' ; c=c+1 ;`
   a. c vaut `'A'`
   b. c vaut `'a'`
   c. Un message d'erreur s'affiche
   d. c vaut `'b'`

4. `if (a%2) printf("bonjour") ;`
   a. N'affiche rien (quelque soit le type et la valeur de a)
   b. Déclenche le message d'erreur `invalid lvalue in assignment`
   c. Affiche bonjour quand a est un entier pair
   d. Affiche bonjour quand a est un entier impair

5. `gcc` est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un mot-clef du langage C
   c. Un identificateur du langage C
   d. Un opérateur du langage C

6. L'expression `(0 == 7%3) && (0 == 9%3)`
   a. entraîne l'affichage d'un message d'erreur
   b. n'a pas de valeur
   c. a pour valeur VRAI
   d. a pour valeur FAUX

7. L'instruction
```
for(i=1 ; i<10 ; i++){
    printf("%i ", i) ;
    printf("%i ", i*i) ;
}
```
   a. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré
   b. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   c. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   d. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré

8. `printf("%c", 'A')`
   a. Affiche le code ASCII du caractère `'A'`
   b. Affiche le code ASCII du caractère stocké dans la variable A
   c. Affiche le caractère dont le code ASCII est stocké dans la variable A
   d. Affiche le caractère `'A'`

9. `if (a%2 == 0) printf("bonjour") ;`
   a. N'affiche rien (quelque soit la valeur de a)
   b. Affiche bonjour quand a est un entier pair
   c. Affiche bonjour quand a est un entier impair
   d. Déclenche le message d'erreur `invalid lvalue in assignment`

10. L'expression `a<=1`
   a. A pour valeur VRAI si $a \leq 1$ et FAUX sinon
   b. Utilise les opérateurs `<` et `=`
   c. Diminue de 1 la valeur de a
   d. Réalise une affectation

11. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
   a. il est impossible de rajouter } tout en gardant le même comportement
   b. après la dernière ligne }
   c. à la fin de la ligne `printf("\ n") ;`
   d. à la fin de la ligne `printf("%i", tab[i]) ;`

12. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
   a. Il y a une faute de syntaxe
   b. `%` est un opérateur permettant de calculer un reste
   c. Il y a une faute d'orthographe
   d. On utilise `%i` car a est de type `int`

13. L'instruction
```
for(i=1 ; i<=n ; i++)
    printf("%i", n-i) ;
```
   a. affiche les entiers de 1 à n en ordre croissant
   b. affiche les entiers de 0 à n-1 en ordre décroissant
   c. affiche les entiers de 1 à n en ordre décroissant
   d. affiche les entiers de 0 à n-1 en ordre croissant

14. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un identificateur du langage C
   c. Un opérateur du langage C
   d. Un mot-clef du langage C

15. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
   a. Contiennent des erreurs
   b. Sont bien indentées
   c. Utilisent un opérateur d'incrémentation
   d. Demandent un entier à l'utilisateur si `i` vaut 1

16. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
    a. 100
    b. 110
    c. 101
    d. 111

17. Si on ajoute 1 au nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 101011010101111
    b. 101011010100000
    c. 101011010100110
    d. 101011010101000

18. `if (a<5) printf("Bonjour") ; a=a+1 ;`
    a. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
    b. Augmente la valeur de a quelque soit a
    c. Affiche bonjour quelque soit a
    d. Affiche bonjour et augmente la valeur de a quelque soit a

19. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 1010110101001111
    b. 1010110101010000
    c. 1010110101001110
    d. 1010110101011110

20. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
    a. `a` vaut 5 et `b` a une valeur indéterminée
    b. `a` vaut 5 et `b` a une valeur indéterminée
    c. `a` vaut 5 et `b` vaut 5
    d. `a` a une valeur indéterminée et `b` vaut 5

21. Après `int t[10] ;`
    a. `t` est un tableau de dix cases numérotées de 1 à 10
    b. `t` est un entier compris entre 1 et 10
    c. `t` est un entier compris entre 0 et 9
    d. `t` est un tableau de dix cases numérotées de 0 à 9

22. En typographie le caractère `&` s'appelle :
    a. L'arpette
    b. L'espagnolette
    c. L'esperluette
    d. La luette

23. Le nombre qui se note 110101 en base 2 se note en base 10 :
    a. 53
    b. 35
    c. 36
    d. 63

24. `if` est :
    a. Un identificateur du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un mot-clef du langage C
    d. Un opérateur du langage C

25. En langage C, #include<stdio.h> permet
    a. d'accélérer la compilation
    b. est obligatoire dans tout programme de langage C
    c. d'utiliser des fonctions d'entrée-sortie
    d. d'accélérer l'exécution

26. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
    a. à la compilation, un message d'erreur s'affichera
    b. à l'exécution, 1 s'affichera
    c. à l'exécution, 0 s'affichera
    d. à l'exécution, n'importe quel nombre s'affichera

27. `printf("%c", A)`
    a. Affiche le code ASCII du caractère stocké dans la variable A
    b. Affiche le code ASCII du caractère `'A'`
    c. Affiche le caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

28. En langage C, une fonction qui ne renvoie rien, c'est :
    a. une fonction de type `void`
    b. une fonction de type `int`
    c. impossible
    d. une fonction de type pointeur

29. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
    a. Un opérateur du langage C
    b. Un mot-clef du langage C
    c. Une commande qu'on tape dans la fenêtre de commande
    d. Un identificateur du langage C

30. `if (a%2 = 0) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier impair
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. N'affiche rien (quelque soit la valeur de a)
    d. Affiche bonjour quand a est un entier pair

31. `=` permet en langage C de :
    a. réaliser une affectation
    b. tester une égalité
    c. comparer deux identificateurs
    d. convertir un int en float

32. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```

a. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
b. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré
c. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
d. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré

33. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
a. $\lfloor \log_2(n) - 1 \rfloor$
b. $\lfloor \log_2(n) \rfloor + 1$
c. $\lfloor \log_2(n) \rfloor - 1$
d. $\lfloor \log_2(n) \rfloor$

34. En langage C, `++` est
a. Un opérateur de décrémentation
b. Un opérateur d'accrétion
c. Un opérateur de décrétinisation
d. Un opérateur d'incrémentation

35. `printf("%i", 'A')`
a. Affiche le caractère dont le code ASCII est stocké dans la variable A
b. Affiche le code ASCII du caractère stocké dans la variable A
c. Affiche le caractère `'A'`
d. Affiche le code ASCII du caractère `'A'`

36. Après `int a, b ; a=3/2 ; b=3%2 ;`
a. `a` vaut 1 et `b` vaut 1
b. `a` vaut 1 et `b` vaut 0
c. `a` vaut 1.5 et `b` vaut 1
d. `a` vaut 1.5 et `b` vaut 0

37. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
a. le i^{e} nombre de Fibonacci à la case n$^{\text{o}}$ i
b. $i!$ à la case n$^{\text{o}}$ i
c. $2i$ à la case n$^{\text{o}}$ i
d. 2^i à la case n$^{\text{o}}$ i

38. On définit une fonction par
```
int f(int x){
    return 3*x+1 ;
}
```
Dans main(), l'expression f(n)
a. est égale à VRAI ou FAUX
b. est logiquement équivalente à 3*n+1
c. a pour valeur 3*n+1
d. affiche 3*n+1

39. La ligne de programme `scanf("%i", &a);`
    a. Permet de stocker dans `a` un entier au choix de l'utilisateur
    b. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
    c. Contient une erreur
    d. Affiche l'adresse mémoire de la variable `a`

40. La ligne de programme `printf("%i, %f, %c", a, b, c);`
    a. Affiche l'entier `a`, le réel `b` et le caractère `c`
    b. Affiche les nombres `a, b, c` sous trois formats différents
    c. Affiche les caractères `'i', 'f', 'c'`
    d. Contient une erreur

Sujet n^o 5

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. Le nombre qui se note 110101 en base 2 se note en base 10 :
   a. 35
   b. 53
   c. 63
   d. 36

2. Après `char c ; c='a' ; c=c+1 ;`
   a. Un message d'erreur s'affiche
   b. c vaut `'b'`
   c. c vaut `'A'`
   d. c vaut `'a'`

3. L'instruction
```
for(i=1 ; i<=n ; i++)
    printf("%i", n-i) ;
```
   a. affiche les entiers de 0 à n-1 en ordre décroissant
   b. affiche les entiers de 1 à n en ordre décroissant
   c. affiche les entiers de 0 à n-1 en ordre croissant
   d. affiche les entiers de 1 à n en ordre croissant

4. Si on ajoute 1 au nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 101011010101000
   b. 101011010100110
   c. 101011010101111
   d. 101011010100000

5. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
   a. 100
   b. 111
   c. 101
   d. 110

6. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
   a. Sont bien indentées
   b. Demandent un entier à l'utilisateur si `i` vaut 1
   c. Utilisent un opérateur d'incrémentation
   d. Contiennent des erreurs

7. En langage C, #include<stdio.h> permet
   a. d'accélérer la compilation
   b. est obligatoire dans tout programme de langage C
   c. d'accélérer l'exécution
   d. d'utiliser des fonctions d'entrée-sortie

8. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 1010110101010000
   b. 1010110101001110
   c. 1010110101011110
   d. 1010110101001111

9. `if (a%2) printf("bonjour") ;`
   a. Déclenche le message d'erreur `invalid lvalue in assignment`
   b. Affiche bonjour quand a est un entier pair
   c. N'affiche rien (quelque soit le type et la valeur de a)
   d. Affiche bonjour quand a est un entier impair

10. `gcc` est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un mot-clef du langage C
   c. Un opérateur du langage C
   d. Un identificateur du langage C

11. En langage C, `++` est
   a. Un opérateur d'incrémentation
   b. Un opérateur de décrémentation
   c. Un opérateur de décrétinisation
   d. Un opérateur d'accrétion

12. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un mot-clef du langage C
   c. Un opérateur du langage C
   d. Un identificateur du langage C

13. L'expression `(0 == 7%3) && (0 == 9%3)`
   a. entraîne l'affichage d'un message d'erreur
   b. n'a pas de valeur
   c. a pour valeur FAUX
   d. a pour valeur VRAI

14. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
   a. $i!$ à la case n$^{\text{o}}$ i
   b. le i^{e} nombre de Fibonacci à la case n$^{\text{o}}$ i
   c. $2i$ à la case n$^{\text{o}}$ i
   d. 2^i à la case n$^{\text{o}}$ i

15. `printf("%i", A)`
   a. Affiche le code ASCII du caractère stocké dans la variable A
   b. Affiche le caractère dont le code ASCII est stocké dans la variable A
   c. Affiche le code ASCII du caractère `'A'`
   d. Affiche le caractère `'A'`

16. On définit une fonction par
```
int f(int x){
    return 3*x+1 ;
}
```
Dans main(), l'expression f(n)
   a. est logiquement équivalente à 3*n+1
   b. est égale à VRAI ou FAUX
   c. affiche 3*n+1
   d. a pour valeur 3*n+1

17. `printf("%c", 'A')`
   a. Affiche le caractère `'A'`
   b. Affiche le caractère dont le code ASCII est stocké dans la variable A
   c. Affiche le code ASCII du caractère `'A'`
   d. Affiche le code ASCII du caractère stocké dans la variable A

18. `printf("%i", 'A')`
   a. Affiche le code ASCII du caractère stocké dans la variable A
   b. Affiche le code ASCII du caractère `'A'`
   c. Affiche le caractère `'A'`
   d. Affiche le caractère dont le code ASCII est stocké dans la variable A

19. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```
   a. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
   b. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré
   c. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
   d. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré

20. L'expression `a<=1`
   a. Diminue de 1 la valeur de a
   b. Réalise une affectation
   c. A pour valeur VRAI si $a \leq 1$ et FAUX sinon
   d. Utilise les opérateurs `<` et `=`

21. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
   a. à l'exécution, n'importe quel nombre s'affichera
   b. à l'exécution, 0 s'affichera
   c. à l'exécution, 1 s'affichera
   d. à la compilation, un message d'erreur s'affichera

22. En langage C, une fonction qui ne renvoie rien, c'est :
   a. une fonction de type `void`
   b. impossible
   c. une fonction de type `int`
   d. une fonction de type pointeur

23. En typographie le caractère `&` s'appelle :
    a. La luette
    b. L'espagnolette
    c. L'esperluette
    d. L'arpette

24. `printf("%c", A)`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le code ASCII du caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

25. L'instruction
```
for(i=1 ; i<10 ; i++){
    printf("%i ", i) ;
    printf("%i ", i*i) ;
}
```
    a. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
    b. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
    c. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré
    d. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré

26. `if (a%2 = 0) printf("bonjour") ;`
    a. N'affiche rien (quelque soit la valeur de a)
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. Affiche bonjour quand a est un entier pair
    d. Affiche bonjour quand a est un entier impair

27. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
    a. $\lfloor \log_2(n) \rfloor - 1$
    b. $\lfloor \log_2(n) \rfloor$
    c. $\lfloor \log_2(n) - 1 \rfloor$
    d. $\lfloor \log_2(n) \rfloor + 1$

28. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
    a. `%` est un opérateur permettant de calculer un reste
    b. On utilise `%i` car a est de type `int`
    c. Il y a une faute d'orthographe
    d. Il y a une faute de syntaxe

29. `=` permet en langage C de :
    a. convertir un int en float
    b. comparer deux identificateurs
    c. tester une égalité
    d. réaliser une affectation

30. Dans le programme de l'exercice 1, `MAX`
    a. est de type `int`
    b. est remplacé par 10 partout où il apparaît lors de la compilation
    c. est de type `float`
    d. est un pointeur

31. `if (a%2 == 0) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier pair
    b. Affiche bonjour quand a est un entier impair
    c. N'affiche rien (quelque soit la valeur de a)
    d. Déclenche le message d'erreur `invalid lvalue in assignment`

32. La ligne de programme `scanf("%i", &a) ;`
    a. Contient une erreur
    b. Affiche l'adresse mémoire de la variable `a`
    c. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
    d. Permet de stocker dans `a` un entier au choix de l'utilisateur

33. `if (a<5) printf("Bonjour") ; a=a+1 ;`
    a. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
    b. Augmente la valeur de a quelque soit a
    c. Affiche bonjour quelque soit a
    d. Affiche bonjour et augmente la valeur de a quelque soit a

34. `if` est :
    a. Un mot-clef du langage C
    b. Un opérateur du langage C
    c. Un identificateur du langage C
    d. Une commande qu'on tape dans la fenêtre de commande

35. Après `int a, b ; a=3/2 ; b=3%2 ;`
    a. `a` vaut 1.5 et `b` vaut 0
    b. `a` vaut 1.5 et `b` vaut 1
    c. `a` vaut 1 et `b` vaut 0
    d. `a` vaut 1 et `b` vaut 1

36. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
    a. Un identificateur du langage C
    b. Un mot-clef du langage C
    c. Un opérateur du langage C
    d. Une commande qu'on tape dans la fenêtre de commande

37. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
    a. `a` a une valeur indéterminée et `b` vaut 5
    b. `a` vaut 5 et `b` a une valeur indéterminée
    c. `a` vaut 5 et `b` vaut 5
    d. `a` vaut 5 et `b` a une valeur indéterminée

38. La ligne de programme `printf("%i, %f, %c", a, b, c) ;`
    a. Affiche les caractères `'i', 'f', 'c'`
    b. Contient une erreur
    c. Affiche les nombres `a, b, c` sous trois formats différents
    d. Affiche l'entier `a`, le réel `b` et le caractère `c`

39. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?

a. après la dernière ligne }

b. à la fin de la ligne `printf("\ n") ;`

c. à la fin de la ligne `printf("%i", tab[i]) ;`

d. il est impossible de rajouter } tout en gardant le même comportement

40. Après `int t[10] ;`

a. `t` est un entier compris entre 0 et 9

b. `t` est un tableau de dix cases numérotées de 0 à 9

c. `t` est un tableau de dix cases numérotées de 1 à 10

d. `t` est un entier compris entre 1 et 10

Sujet n^o 6

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. L'expression `a<=1`
   a. Réalise une affectation
   b. Utilise les opérateurs `<` et `=`
   c. Diminue de 1 la valeur de a
   d. A pour valeur VRAI si $a \leq 1$ et FAUX sinon

2. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
   a. Demandent un entier à l'utilisateur si `i` vaut 1
   b. Sont bien indentées
   c. Utilisent un opérateur d'incrémentation
   d. Contiennent des erreurs

3. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
   a. Un mot-clef du langage C
   b. Un opérateur du langage C
   c. Une commande qu'on tape dans la fenêtre de commande
   d. Un identificateur du langage C

4. `if (a<5) printf("Bonjour") ; a=a+1 ;`
   a. Affiche bonjour et augmente la valeur de a quelque soit a
   b. Augmente la valeur de a quelque soit a
   c. Affiche bonjour quelque soit a
   d. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a

5. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
   a. après la dernière ligne }
   b. à la fin de la ligne `printf("\ n") ;`
   c. il est impossible de rajouter } tout en gardant le même comportement
   d. à la fin de la ligne `printf("%i", tab[i]) ;`

6. En typographie le caractère `&` s'appelle :
   a. L'espagnolette
   b. L'arpette
   c. L'esperluette
   d. La luette

7. Après `int t[10] ;`
   a. `t` est un entier compris entre 0 et 9
   b. `t` est un tableau de dix cases numérotées de 1 à 10
   c. `t` est un tableau de dix cases numérotées de 0 à 9
   d. `t` est un entier compris entre 1 et 10

8. Après `char c ; c='a' ; c=c+1 ;`
   a. c vaut `'b'`
   b. c vaut `'A'`
   c. Un message d'erreur s'affiche
   d. c vaut `'a'`

9. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
   a. $2i$ à la case n$^{\text{o}}$ i
   b. le i^{e} nombre de Fibonacci à la case n$^{\text{o}}$ i
   c. 2^i à la case n$^{\text{o}}$ i
   d. $i!$ à la case n$^{\text{o}}$ i

10. L'expression `(0 == 7%3) && (0 == 9%3)`
   a. a pour valeur VRAI
   b. entraîne l'affichage d'un message d'erreur
   c. a pour valeur FAUX
   d. n'a pas de valeur

11. `=` permet en langage C de :
   a. convertir un int en float
   b. réaliser une affectation
   c. tester une égalité
   d. comparer deux identificateurs

12. Si on ajoute 1 au nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 101011010100110
   b. 101011010101000
   c. 101011010101111
   d. 101011010100000

13. Le nombre qui se note 110101 en base 2 se note en base 10 :
   a. 36
   b. 53
   c. 35
   d. 63

14. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
   a. 1010110101001111
   b. 1010110101010000
   c. 1010110101001110
   d. 1010110101011110

15. `if` est :
   a. Un opérateur du langage C
   b. Un mot-clef du langage C
   c. Une commande qu'on tape dans la fenêtre de commande
   d. Un identificateur du langage C

16. En langage C, une fonction qui ne renvoie rien, c'est :
    a. une fonction de type `int`
    b. une fonction de type pointeur
    c. une fonction de type `void`
    d. impossible

17. `if (a%2 = 0) printf("bonjour") ;`
    a. N'affiche rien (quelque soit la valeur de a)
    b. Affiche bonjour quand a est un entier impair
    c. Déclenche le message d'erreur `invalid lvalue in assignment`
    d. Affiche bonjour quand a est un entier pair

18. En langage C, `++` est
    a. Un opérateur d'accrétion
    b. Un opérateur de décrétinisation
    c. Un opérateur de décrémentation
    d. Un opérateur d'incrémentation

19. `printf("%c", 'A')`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère `'A'`
    c. Affiche le caractère dont le code ASCII est stocké dans la variable A
    d. Affiche le code ASCII du caractère stocké dans la variable A

20. On définit une fonction par
```
int f(int x){
    return 3*x+1 ;
}
```
Dans main(), l'expression f(n)
    a. est égale à VRAI ou FAUX
    b. affiche 3*n+1
    c. est logiquement équivalente à 3*n+1
    d. a pour valeur 3*n+1

21. `printf("%i", 'A')`
    a. Affiche le caractère `'A'`
    b. Affiche le caractère dont le code ASCII est stocké dans la variable A
    c. Affiche le code ASCII du caractère stocké dans la variable A
    d. Affiche le code ASCII du caractère `'A'`

22. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
    a. à l'exécution, n'importe quel nombre s'affichera
    b. à l'exécution, 0 s'affichera
    c. à l'exécution, 1 s'affichera
    d. à la compilation, un message d'erreur s'affichera

23. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
    a. $\lfloor \log_2(n) \rfloor + 1$
    b. $\lfloor \log_2(n) \rfloor$
    c. $\lfloor \log_2(n) - 1 \rfloor$
    d. $\lfloor \log_2(n) \rfloor - 1$

24. `if (a%2 == 0) printf("bonjour") ;`
    a. N'affiche rien (quelque soit la valeur de a)
    b. Affiche bonjour quand a est un entier pair
    c. Déclenche le message d'erreur `invalid lvalue in assignment`
    d. Affiche bonjour quand a est un entier impair

25. `gcc` est :
    a. Un mot-clef du langage C
    b. Un identificateur du langage C
    c. Une commande qu'on tape dans la fenêtre de commande
    d. Un opérateur du langage C

26. La ligne de programme `scanf("%i", &a) ;`
    a. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
    b. Affiche l'adresse mémoire de la variable `a`
    c. Contient une erreur
    d. Permet de stocker dans `a` un entier au choix de l'utilisateur

27. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
    a. Il y a une faute de syntaxe
    b. Il y a une faute d'orthographe
    c. `%` est un opérateur permettant de calculer un reste
    d. On utilise `%i` car a est de type `int`

28. Après `int a, b ; a=3/2 ; b=3%2 ;`
    a. `a` vaut 1.5 et `b` vaut 1
    b. `a` vaut 1.5 et `b` vaut 0
    c. `a` vaut 1 et `b` vaut 1
    d. `a` vaut 1 et `b` vaut 0

29. En langage C, #include<stdio.h> permet
    a. est obligatoire dans tout programme de langage C
    b. d'accélérer l'exécution
    c. d'accélérer la compilation
    d. d'utiliser des fonctions d'entrée-sortie

30. L'instruction
```
for(i=1 ; i<10 ; i++){
    printf("%i ", i) ;
    printf("%i ", i*i) ;
}
```
    a. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré
    b. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré
    c. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
    d. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100

31. La ligne de programme `printf("%i, %f, %c", a, b, c) ;`
    a. Contient une erreur
    b. Affiche l'entier `a`, le réel `b` et le caractère `c`
    c. Affiche les nombres `a, b, c` sous trois formats différents
    d. Affiche les caractères `'i', 'f', 'c'`

32. Dans le programme de l'exercice 1, `MAX`
    a. est un pointeur
    b. est de type `float`
    c. est remplacé par 10 partout où il apparaît lors de la compilation
    d. est de type `int`

33. L'instruction
```
for(i=1 ; i<=n ; i++)
        printf("%i", n-i) ;
```
    a. affiche les entiers de 1 à n en ordre décroissant
    b. affiche les entiers de 0 à n-1 en ordre croissant
    c. affiche les entiers de 0 à n-1 en ordre décroissant
    d. affiche les entiers de 1 à n en ordre croissant

34. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
    a. 111
    b. 100
    c. 110
    d. 101

35. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
    a. Une commande qu'on tape dans la fenêtre de commande
    b. Un opérateur du langage C
    c. Un mot-clef du langage C
    d. Un identificateur du langage C

36. `printf("%c", A)`
    a. Affiche le caractère dont le code ASCII est stocké dans la variable A
    b. Affiche le caractère `'A'`
    c. Affiche le code ASCII du caractère stocké dans la variable A
    d. Affiche le code ASCII du caractère `'A'`

37. Les instructions
```
for(i=1 ; i<10 ; i++)
        printf("%i ", i) ;
        printf("%i ", i*i) ;
```
    a. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré
    b. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré
    c. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
    d. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100

38. `printf("%i", A)`
    a. Affiche le code ASCII du caractère stocké dans la variable A
    b. Affiche le code ASCII du caractère `'A'`
    c. Affiche le caractère dont le code ASCII est stocké dans la variable A
    d. Affiche le caractère `'A'`

39. `if (a%2) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier impair
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. Affiche bonjour quand a est un entier pair
    d. N'affiche rien (quelque soit le type et la valeur de a)

40. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
    a. `a` vaut 5 et `b` a une valeur indéterminée
    b. `a` vaut 5 et `b` vaut 5
    c. `a` a une valeur indéterminée et `b` vaut 5
    d. `a` vaut 5 et `b` a une valeur indéterminée

Sujet n^o 7

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. Dans le programme de l'exercice 1, `MAX`
   a. est remplacé par 10 partout où il apparaît lors de la compilation
   b. est un pointeur
   c. est de type `int`
   d. est de type `float`

2. La ligne de programme `printf("%i, %f, %c", a, b, c) ;`
   a. Affiche les nombres `a, b, c` sous trois formats différents
   b. Contient une erreur
   c. Affiche l'entier `a`, le réel `b` et le caractère `c`
   d. Affiche les caractères `'i', 'f', 'c'`

3. L'instruction
```
for(i=1 ; i<10 ; i++){
    printf("%i ", i) ;
    printf("%i ", i*i) ;
}
```
   a. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré
   b. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   c. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré
   d. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100

4. En langage C, #include<stdio.h> permet
   a. est obligatoire dans tout programme de langage C
   b. d'accélérer l'exécution
   c. d'utiliser des fonctions d'entrée-sortie
   d. d'accélérer la compilation

5. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
   a. Un identificateur du langage C
   b. Un opérateur du langage C
   c. Un mot-clef du langage C
   d. Une commande qu'on tape dans la fenêtre de commande

6. En langage C, une fonction qui ne renvoie rien, c'est :
   a. une fonction de type pointeur
   b. impossible
   c. une fonction de type `int`
   d. une fonction de type `void`

7. `=` permet en langage C de :
   a. réaliser une affectation
   b. convertir un int en float
   c. comparer deux identificateurs
   d. tester une égalité

8. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```

a. Demandent un entier à l'utilisateur si `i` vaut 1
b. Contiennent des erreurs
c. Sont bien indentées
d. Utilisent un opérateur d'incrémentation

9. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
a. `a` vaut 5 et `b` a une valeur indéterminée
b. `a` vaut 5 et `b` vaut 5
c. `a` vaut 5 et `b` a une valeur indéterminée
d. `a` a une valeur indéterminée et `b` vaut 5

10. En typographie le caractère `&` s'appelle :
a. L'esperluette
b. La luette
c. L'arpette
d. L'espagnolette

11. On définit une fonction par
```
int f(int x){
  return 3*x+1 ;
}
```
.
Dans main(), l'expression f(n)
a. affiche 3*n+1
b. est logiquement équivalente à 3*n+1
c. est égale à VRAI ou FAUX
d. a pour valeur 3*n+1

12. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
a. $\lfloor \log_2(n) \rfloor - 1$
b. $\lfloor \log_2(n) \rfloor + 1$
c. $\lfloor \log_2(n) \rfloor$
d. $\lfloor \log_2(n) - 1 \rfloor$

13. `printf("%c", 'A')`
a. Affiche le code ASCII du caractère stocké dans la variable A
b. Affiche le code ASCII du caractère `'A'`
c. Affiche le caractère dont le code ASCII est stocké dans la variable A
d. Affiche le caractère `'A'`

14. Après `int t[10] ;`
a. `t` est un tableau de dix cases numérotées de 0 à 9
b. `t` est un tableau de dix cases numérotées de 1 à 10
c. `t` est un entier compris entre 0 et 9
d. `t` est un entier compris entre 1 et 10

15. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
    a. Il y a une faute d'orthographe
    b. On utilise `%i` car a est de type `int`
    c. Il y a une faute de syntaxe
    d. `%` est un opérateur permettant de calculer un reste

16. `if (a%2 == 0) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier impair
    b. Déclenche le message d'erreur `invalid lvalue in assignment`
    c. N'affiche rien (quelque soit la valeur de a)
    d. Affiche bonjour quand a est un entier pair

17. L'expression `(0 == 7%3) && (0 == 9%3)`
    a. entraîne l'affichage d'un message d'erreur
    b. a pour valeur VRAI
    c. a pour valeur FAUX
    d. n'a pas de valeur

18. Après `int a, b ; a=3/2 ; b=3%2 ;`
    a. `a` vaut 1 et `b` vaut 1
    b. `a` vaut 1 et `b` vaut 0
    c. `a` vaut 1.5 et `b` vaut 0
    d. `a` vaut 1.5 et `b` vaut 1

19. `if (a%2 = 0) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier pair
    b. Affiche bonjour quand a est un entier impair
    c. Déclenche le message d'erreur `invalid lvalue in assignment`
    d. N'affiche rien (quelque soit la valeur de a)

20. `printf("%i", A)`
    a. Affiche le code ASCII du caractère stocké dans la variable A
    b. Affiche le caractère `'A'`
    c. Affiche le caractère dont le code ASCII est stocké dans la variable A
    d. Affiche le code ASCII du caractère `'A'`

21. Si on ajoute 1 au nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 101011010101111
    b. 101011010101000
    c. 101011010100000
    d. 101011010100110

22. Le nombre qui se note 110101 en base 2 se note en base 10 :
    a. 35
    b. 63
    c. 53
    d. 36

23. `if (a<5) printf("Bonjour") ; a=a+1 ;`
    a. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
    b. Augmente la valeur de a quelque soit a
    c. Affiche bonjour et augmente la valeur de a quelque soit a
    d. Affiche bonjour quelque soit a

24. `if (a%2) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier pair
    b. Affiche bonjour quand a est un entier impair
    c. Déclenche le message d'erreur `invalid lvalue in assignment`
    d. N'affiche rien (quelque soit le type et la valeur de a)

25. `gcc` est :
    a. Un mot-clef du langage C
    b. Un opérateur du langage C
    c. Un identificateur du langage C
    d. Une commande qu'on tape dans la fenêtre de commande

26. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
    a. après la dernière ligne }
    b. il est impossible de rajouter } tout en gardant le même comportement
    c. à la fin de la ligne `printf("\ n") ;`
    d. à la fin de la ligne `printf("%i", tab[i]) ;`

27. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
    a. Un mot-clef du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un opérateur du langage C
    d. Un identificateur du langage C

28. `printf("%i", 'A')`
    a. Affiche le caractère `'A'`
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le caractère dont le code ASCII est stocké dans la variable A
    d. Affiche le code ASCII du caractère `'A'`

29. Après `char c ; c='a' ; c=c+1 ;`
    a. c vaut `'b'`
    b. c vaut `'a'`
    c. c vaut `'A'`
    d. Un message d'erreur s'affiche

30. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```
    a. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
    b. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
    c. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré
    d. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré

31. `printf("%c", A)`
    a. Affiche le code ASCII du caractère `'A'`
    b. Affiche le code ASCII du caractère stocké dans la variable A
    c. Affiche le caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

32. L'instruction
```
for(i=1 ; i<=n ; i++)
        printf("%i", n-i) ;
```
    a. affiche les entiers de 1 à n en ordre décroissant
    b. affiche les entiers de 0 à n-1 en ordre croissant
    c. affiche les entiers de 0 à n-1 en ordre décroissant
    d. affiche les entiers de 1 à n en ordre croissant

33. L'expression `a<=1`
    a. Utilise les opérateurs `<` et `=`
    b. Diminue de 1 la valeur de a
    c. Réalise une affectation
    d. A pour valeur VRAI si $a \leq 1$ et FAUX sinon

34. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 1010110101001110
    b. 1010110101010000
    c. 1010110101001111
    d. 1010110101011110

35. `if` est :
    a. Un mot-clef du langage C
    b. Une commande qu'on tape dans la fenêtre de commande
    c. Un opérateur du langage C
    d. Un identificateur du langage C

36. La ligne de programme `scanf("%i", &a) ;`
    a. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
    b. Contient une erreur
    c. Affiche l'adresse mémoire de la variable `a`
    d. Permet de stocker dans `a` un entier au choix de l'utilisateur

37. En langage C, `++` est
    a. Un opérateur d'accrétion
    b. Un opérateur de décrétinisation
    c. Un opérateur de décrémentation
    d. Un opérateur d'incrémentation

38. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
    a. à l'exécution, n'importe quel nombre s'affichera
    b. à l'exécution, 1 s'affichera
    c. à l'exécution, 0 s'affichera
    d. à la compilation, un message d'erreur s'affichera

39. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`

a. $2i$ à la case n$^{\text{o}}$ i
b. $i!$ à la case n$^{\text{o}}$ i
c. 2^i à la case n$^{\text{o}}$ i
d. le i^{e} nombre de Fibonacci à la case n$^{\text{o}}$ i

40. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :

a. 110
b. 111
c. 101
d. 100

Sujet n^o 8

Langage C
Licence MASS 1re année
1er semestre

Examen janvier 2007
Année 2006–2007
Documents et calculatrices interdits

ATTENTION : pour le QCM, rendre exclusivement la feuille réponse jointe au sujet, que vous glisserez dans votre copie. Pour limiter le risque de perte, pour préserver l'anonymat et pour faciliter la correction : il est IMPÉRATIF de recopier le numéro du sujet sur votre copie, et sur la feuille réponse du QCM.

Avertissement : certaines questions du QCM font référence à l'exercice 1. Il est donc préférable de faire l'exercice 1 avant le QCM.

Exercice 1

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}

return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

Exercice 2

1. Écrire une fonction de prototype `void rempli(int t[], int n)` qui stocke dans les cases 0 à `n` du tableau `t` les carrés des entiers. Par exemple, à la case `t[3]` on doit avoir l'entier `9`.

2. Écrire une fonction de prototype `int indice(int t[], int n)` qui renvoie un indice i tel que `t[i]` ait pour valeur `n` si un tel entier existe. Sinon, la fonction renvoie -1.

Exercice 3

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$, $u_1 \in \mathbb{R}$ et pour tout $n \geq 2$, $u_n = (u_{n-1} - u_{n-2})/2$.

1. Écrire un programme qui demande à l'utilisateur des réels a, b et un entier n, puis qui calcule et affiche u_n avec $u_0 = a$ et $u_1 = b$.

2. Ajouter au programme précédent quelques lignes pour que le programme affiche en plus le premier terme u_k telle que $u_k - u_{k-1} < 0.001$. Si un tel entier n'existe pas, ou si sa taille dépasse les capacités de l'ordinateur, vous êtes autorisé à donner un programme qui boucle ou donne une mauvaise réponse.

Exercice 4

Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 5

QCM. Rappel important : voir en début de sujet le mode d'emploi pour répondre au QCM

1. Dans la ligne de programme `printf("Le nombre vaut %i.", a) ;`
   a. Il y a une faute d'orthographe
   b. Il y a une faute de syntaxe
   c. `%` est un opérateur permettant de calculer un reste
   d. On utilise `%i` car a est de type `int`

2. `printf("%i", 'A')`
   a. Affiche le caractère `'A'`
   b. Affiche le code ASCII du caractère stocké dans la variable A
   c. Affiche le caractère dont le code ASCII est stocké dans la variable A
   d. Affiche le code ASCII du caractère `'A'`

3. Les instructions
```
for(i=1 ; i<10 ; i++)
    printf("%i ", i) ;
    printf("%i ", i*i) ;
```
   a. Sont mal indentées et affichent les 9 premiers entiers ainsi que leur carré
   b. Sont bien indentées et affichent les 9 premiers entiers ainsi que leur carré
   c. Sont mal indentées, affichent les 9 premiers entiers, puis affichent le nombre 100
   d. Sont bien indentées, affichent les 9 premiers entiers, puis affichent le nombre 100

4. `if` est :
   a. Une commande qu'on tape dans la fenêtre de commande
   b. Un mot-clef du langage C
   c. Un identificateur du langage C
   d. Un opérateur du langage C

5. `printf("%c", A)`
   a. Affiche le caractère `'A'`
   b. Affiche le code ASCII du caractère stocké dans la variable A
   c. Affiche le code ASCII du caractère `'A'`
   d. Affiche le caractère dont le code ASCII est stocké dans la variable A

6. On rappelle que $\lfloor x \rfloor$ désigne le nombre entier immédiatement inférieur au nombre réel x. Si n est un nombre entier, alors le nombre de chiffres de n en base 2 est égal à :
   a. $\lfloor \log_2(n) - 1 \rfloor$
   b. $\lfloor \log_2(n) \rfloor - 1$
   c. $\lfloor \log_2(n) \rfloor + 1$
   d. $\lfloor \log_2(n) \rfloor$

7. En langage C, une fonction qui ne renvoie rien, c'est :
   a. une fonction de type `void`
   b. impossible
   c. une fonction de type pointeur
   d. une fonction de type `int`

8. En typographie le caractère `&` s'appelle :
   a. La luette
   b. L'arpette
   c. L'espagnolette
   d. L'esperluette

9. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
   a. Un identificateur du langage C
   b. Une commande qu'on tape dans la fenêtre de commande
   c. Un mot-clef du langage C
   d. Un opérateur du langage C

10. `if (a%2 = 0) printf("bonjour") ;`
   a. N'affiche rien (quelque soit la valeur de a)
   b. Déclenche le message d'erreur `invalid lvalue in assignment`
   c. Affiche bonjour quand a est un entier impair
   d. Affiche bonjour quand a est un entier pair

11. Les instructions
```
int i ;
if i=1
  scanf("%i", i) ;
  printf("%i", i, i*i) ;
```
   a. Contiennent des erreurs
   b. Demandent un entier à l'utilisateur si `i` vaut 1
   c. Utilisent un opérateur d'incrémentation
   d. Sont bien indentées

12. `if (a%2 == 0) printf("bonjour") ;`
   a. Affiche bonjour quand a est un entier impair
   b. Déclenche le message d'erreur `invalid lvalue in assignment`
   c. N'affiche rien (quelque soit la valeur de a)
   d. Affiche bonjour quand a est un entier pair

13. L'instruction
```
for(i=1 ; i<10 ; i++){
  printf("%i ", i) ;
  printf("%i ", i*i) ;
}
```
   a. Est bien indentée et affiche les 9 premiers entiers ainsi que leur carré
   b. Est mal indentée, affiche les 9 premiers entiers, puis affiche le nombre 100
   c. Est mal indentée et affiche les 9 premiers entiers ainsi que leur carré
   d. Est bien indentée, affiche les 9 premiers entiers, puis affiche le nombre 100

14. L'expression `a<=1`
   a. Utilise les opérateurs `<` et `=`
   b. Réalise une affectation
   c. A pour valeur VRAI si $a \leq 1$ et FAUX sinon
   d. Diminue de 1 la valeur de a

15. Dans le programme de l'exercice 1, `MAX`
    a. est remplacé par 10 partout où il apparaît lors de la compilation
    b. est un pointeur
    c. est de type `float`
    d. est de type `int`

16. `if (a%2) printf("bonjour") ;`
    a. Affiche bonjour quand a est un entier pair
    b. Affiche bonjour quand a est un entier impair
    c. N'affiche rien (quelque soit le type et la valeur de a)
    d. Déclenche le message d'erreur `invalid lvalue in assignment`

17. Dans le programme de l'exercice 1, on rajoute { à la fin de la ligne `for (i=l-1 ; i>=0 ; i--)`. Où faut-il rajouter } pour que le programme garde le même comportement ?
    a. à la fin de la ligne `printf("\ n") ;`
    b. après la dernière ligne }
    c. à la fin de la ligne `printf("%i", tab[i]) ;`
    d. il est impossible de rajouter } tout en gardant le même comportement

18. `printf("%c", 'A')`
    a. Affiche le code ASCII du caractère stocké dans la variable A
    b. Affiche le caractère `'A'`
    c. Affiche le code ASCII du caractère `'A'`
    d. Affiche le caractère dont le code ASCII est stocké dans la variable A

19. L'instruction `for(i=1 ; i<=n ; i++)`
    `printf("%i", n-i) ;`
    a. affiche les entiers de 0 à n-1 en ordre croissant
    b. affiche les entiers de 1 à n en ordre décroissant
    c. affiche les entiers de 1 à n en ordre croissant
    d. affiche les entiers de 0 à n-1 en ordre décroissant

20. Après `int a, b ; int *x, *y ; a=5 ; x=&a ; b=*x ; a=*y ;`
    a. `a` vaut 5 et `b` a une valeur indéterminée
    b. `a` vaut 5 et `b` vaut 5
    c. `a` a une valeur indéterminée et `b` vaut 5
    d. `a` vaut 5 et `b` a une valeur indéterminée

21. `printf("%i", A)`
    a. Affiche le caractère dont le code ASCII est stocké dans la variable A
    b. Affiche le code ASCII du caractère `'A'`
    c. Affiche le caractère `'A'`
    d. Affiche le code ASCII du caractère stocké dans la variable A

22. La ligne de programme `printf("%i, %f, %c", a, b, c) ;`
    a. Affiche les caractères `'i', 'f', 'c'`
    b. Affiche l'entier `a`, le réel `b` et le caractère `c`
    c. Contient une erreur
    d. Affiche les nombres `a, b, c` sous trois formats différents

23. L'expression `(0 == 7%3) && (0 == 9%3)`
    a. n'a pas de valeur
    b. a pour valeur FAUX
    c. entraîne l'affichage d'un message d'erreur
    d. a pour valeur VRAI

24. La ligne de programme `scanf("%i", &a) ;`
    a. Stocke dans `i` le caractère dont le code ASCII est stocké dans `a`
    b. Affiche l'adresse mémoire de la variable `a`
    c. Contient une erreur
    d. Permet de stocker dans `a` un entier au choix de l'utilisateur

25. `t[0]=1 ; for(i=1 ; i<=10 ; i++) t[i]=2*t[i-1] ;` permet de stocker dans le tableau `t`
    a. $i!$ à la case n^{o} i
    b. $2i$ à la case n^{o} i
    c. 2^i à la case n^{o} i
    d. le i^{e} nombre de Fibonacci à la case n^{o} i

26. Après `char c ; c='a' ; c=c+1 ;`
    a. c vaut `'a'`
    b. c vaut `'A'`
    c. c vaut `'b'`
    d. Un message d'erreur s'affiche

27. `=` permet en langage C de :
    a. convertir un int en float
    b. tester une égalité
    c. comparer deux identificateurs
    d. réaliser une affectation

28. Le nombre qui se note 110101 en base 2 se note en base 10 :
    a. 53
    b. 35
    c. 63
    d. 36

29. En langage C, #include<stdio.h> permet
    a. d'accélérer la compilation
    b. d'utiliser des fonctions d'entrée-sortie
    c. d'accélérer l'exécution
    d. est obligatoire dans tout programme de langage C

30. Dans le programme de l'exercice 1, si on ajoute avant la dernière accolade de `main()` la ligne `printf("%i", c) ;`
    a. à la compilation, un message d'erreur s'affichera
    b. à l'exécution, 1 s'affichera
    c. à l'exécution, 0 s'affichera
    d. à l'exécution, n'importe quel nombre s'affichera

31. `if (a<5) printf("Bonjour") ; a=a+1 ;`
    a. Affiche bonjour et augmente la valeur de a quelque soit a
    b. Affiche bonjour quelque soit a
    c. N'affiche pas bonjour et n'augmente pas la valeur de a quelque soit a
    d. Augmente la valeur de a quelque soit a

32. Dans le programme `main(){int a ; a=3%5 ;}`, `a` est :
    a. Un opérateur du langage C
    b. Un mot-clef du langage C
    c. Une commande qu'on tape dans la fenêtre de commande
    d. Un identificateur du langage C

33. Si on multiplie par 2 le nombre qui se note en base 2 101011010100111, on obtient le nombre qui se note en base 2 :
    a. 1010110101001110
    b. 1010110101010000
    c. 1010110101001111
    d. 1010110101011110

34. Après `int a, b ; a=3/2 ; b=3%2 ;`
    a. `a` vaut 1.5 et `b` vaut 0
    b. `a` vaut 1.5 et `b` vaut 1
    c. `a` vaut 1 et `b` vaut 1
    d. `a` vaut 1 et `b` vaut 0

35. Après `int t[10] ;`
    a. `t` est un tableau de dix cases numérotées de 0 à 9
    b. `t` est un tableau de dix cases numérotées de 1 à 10
    c. `t` est un entier compris entre 1 et 10
    d. `t` est un entier compris entre 0 et 9

36. On rappelle que les trois premiers chiffres de 8 écrit en base 2 sont 100. On écrit en base 2 le nombre 99. Les trois premiers chiffres sont :
    a. 110
    b. 100
    c. 101
    d. 111

37. `gcc` est :
    a. Un identificateur du langage C
    b. Un opérateur du langage C
    c. Un mot-clef du langage C
    d. Une commande qu'on tape dans la fenêtre de commande

38. En langage C, `++` est
    a. Un opérateur d'accrétion
    b. Un opérateur d'incrémentation
    c. Un opérateur de décrétinisation
    d. Un opérateur de décrémentation

39. Si on ajoute 1 au nombre qui se note en base 2 1010110101001111, on obtient le nombre qui se note en base 2 :

a. 1010110101000000
b. 1010110101010000
c. 1010110101011111
d. 1010110101001110

40. On définit une fonction par
```
int f(int x){
  return 3*x+1 ;
}
```
.

Dans main(), l'expression f(n)

a. affiche 3*n+1
b. est égale à VRAI ou FAUX
c. est logiquement équivalente à 3*n+1
d. a pour valeur 3*n+1