

Éléments d'informatique, rattrapage


Durée : 3 heures.

Documents autorisés : Aucun.

Recommandations : Un barème vous est donné à titre indicatif afin de vous permettre de gérer votre temps. La notation prendra en compte à la fois la syntaxe et la sémantique de vos programmes, c'est-à-dire qu'ils doivent compiler correctement. Une fois votre programme écrit, il est recommandé de le faire tourner à la main sur un exemple pour s'assurer de sa correction.

1 Trace d'un programme avec fonctions

Question A. Simulez l'exécution du programme figure 1, en réalisant sa **trace**.

 4 pt
36 min

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* declarations constantes et types utilisateurs */
5
6  /* declarations de fonctions utilisateurs */
7  double foo(int n, double x);
8  double bar(double x);
9
10 /* fonction principale */
11 int main()
12 {
13     int n = 3;
14     double x = 6;
15     double res;
16
17     res = foo(n, x);
18     printf("bar^%d(%lg) = %lg\n", n, x, res);
19
20     return EXIT_SUCCESS;
21 }
22
23 /* definitions de fonctions utilisateurs */
24 double foo(int n, double x)
25 {
26     int i;
27     for (i = 0; i < n; i = i + 1)
28     {
29         x = bar(x);
30     }
31     return x;
32 }
33
34 double bar(double x)
35 {
36     return x / 2.0 + 1.0;
37 }
```

FIGURE 1 – Programme pour la trace

main()

ligne	n	x	res	Aff.						
initialisation	3	6.0	?							
17										
foo(3, 6.0)										
ligne					n	x	i	Aff.		
initialisation					3	6.0	?			
27							0			
29					bar(6.0)					
ligne									x	Aff.
initialisation									6.0	
36					renvoie 4.0					
29						4.0				
30							1			
29					bar(4.0)					
ligne									x	Aff.
initialisation									4.0	
36					renvoie 3.0					
29						3.0				
30							2			
29	bar(3.0)									
ligne					x	Aff.				
initialisation					3.0					
36	renvoie 2.5									
29		2.5								
30			3							
31	renvoie 2.5									
17			2.5							
18				bar^3(6) = 2.5						
20	renvoie EXIT_SUCCESS									

TABLE 1 – Trace du programme de l'exercice 2.

Correction. Table 1 page 2.

Barème. Maximum 4pt. Si des erreurs, maximum 3,5 pt en sommant :

- (a) +1 pt deux premières lignes de la trace du main sont correctes (identification des variables et leurs initialisations).
- * −0,5 pt par variable manquante ou en trop
 - * −0,25 pt par initialisation manquante ou en trop
- (b) +2 pt Pour l'appel à foo :
- * +0,5 pt pour foo(3, 6) ligne 17
 - * +0,5 pt, +0,25 pt par colonne paramètre formel n et x bien initialisées à 3 et 6
 - * +0,5 pt pour le déroulement correct de la boucle sans se soucier de ce qui s'y passe (trois tours, $i = 3$), 0.25 pt si juste oublié de $i = 3$.
 - * +0,5 pt pour le fait que x change de valeur au moins une fois ligne 29.
- (c) +1 pt pour au moins un des (trois) appels à bar correct et dans foo(). Pénalité de −0.5 pt si contient un (ou plusieurs) appel imbriqué.
- * +0,25 pt pour bar(6) ligne 19.
 - * +0,5 pt si retourne un résultat exacte.
- (d) +0,5 pt affichage ligne 18 correct et cohérent avec les valeurs trouvées. 0pt si une erreur.

2 Sans fonctions, for ou while, arbre de décision (11 pt)

Il est demandé de résoudre les trois problèmes suivants sans définir de fonctions utilisateurs et en faisant le meilleur choix entre for et while s'il y a des boucles. L'ensemble du code sera à écrire dans la fonction principale `main`.

2.1 Puissance

Question B. Écrire le programme qui : demande à l'utilisateur d'entrer un nombre à virgule x , puis un entier n ; et qui calcule puis affiche la valeur de x^n , comme dans l'exemple suivant.

2.5 pt
22 min

On supposera que l'entier saisi est toujours supérieur ou égal à zéro (on ne teste pas la saisie de l'utilisateur).

Exemple d'exécution :

```
Saisissez un nombre decimal : 0.5
Saisissez un entier : 3
0.5 puissance 3 = 0.125
```

Correction.

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* declarations constantes et types utilisateurs */
5
6
7  /* fonction principale */
8  int main()
9  {
10     double x = 0;
11     int n = 0;
12     double p = 1; /* accumulateur pour le produit */
13     int i;         /* var de boucle */
14
15     printf("Saisissez un nombre décimal : ");
16     scanf("%lg", &x);
17
18     printf("Saisissez un entier : ");
19     scanf("%d", &n);
20
21     for (i = 0; i < n; i = i + 1)
22     {
23         p = p * x;
24     }
25
26     printf("%lg puissance %d = %lg\n", x, n, p);
27
28     return EXIT_SUCCESS;
29 }
```

Barème. Si tout juste à l'aide d'un for : 2.5 pt sinon (y compris si while) total maximum de 2 pt pris dans les points suivants :


- 0.25 pt déclaration (avec ou sans initialisation) d'une variable double ou float pour la saisie de x
- 0.25 pt déclaration (avec ou sans initialisation) d'une variable entière pour la saisie de n
- 0.25 pt déclaration d'un accumulateur de type double ou float pour stocker le produit
- 0.25 pt initialisation à 1 de l'accumulateur
- 0.25 pt saisie de n à l'aide d'un `scanf` avec ou sans ℓ

- 0.25 pt saisie de n à l'aide d'un `scanf` avec ou sans \mathcal{E}
- 0.5 pt boucle `for` pas de point en moins si oublie de déclaration de la variable de boucle
- 0.25 pt bloc de boucle exécuté n fois
- 0.25 pt affichage du résultat du calcul

2.2 Moins de dette privée

Pippo emprunte 100 brouzoufs à 6% d'intérêts annuel. Il rembourse par versements de 25 brouzoufs chaque année, en commençant un an après la date d'emprunt. Combien d'années devra t'il rembourser son emprunt ? Chaque année, on calcule d'abord les intérêts à ajouter à la dette actuelle, puis on soustrait le montant du versement annuel.

Question C. Écrire un programme qui affiche le plan de remboursement de Pippo, c'est à dire, pour chaque année : les intérêts de l'année, le versement effectué, la dette restant à rembourser.

 2.5 pt
22 min

Votre programme doit pouvoir être exécuté pour n'importe quelle autre initialisation des données du prêt.

Exemple d'exécution :

```
annee 1, interets : 6, versement : 25, dette : 81
annee 2, interets : 4.86, versement : 25, dette : 60.86
annee 3, interets : 3.6516, versement : 25, dette : 39.5116
annee 4, interets : 2.3707, versement : 25, dette : 16.8823
annee 5, interets : 1.01294, versement : 25, dette : -7.10477
```

Correction.


```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* déclaration constantes et types utilisateurs */
5
6  /* Fonction principale */
7  int main()
8  {
9      double dette = 100.0;
10     double versement = 25.0;
11     double taux = 0.06;
12     int annee = 0;
13     double interets;
14
15     while (dette > 0)
16     {
17         interets = dette * taux;
18         dette = dette + interets - versement;
19         annee = annee + 1;
20         printf("annee %d, interets : %g, versement : %g, dette : %g\n",
21             annee, interets, versement, dette);
22     }
23
24     return EXIT_SUCCESS;
25 }
```

Barème. Si tout juste à l'aide d'un `while` : 2.5 pt sinon maxi 2, 25 pt.

- 0.25 pt déclaration et initialisation de la dette et du taux comme des double (on ne regarde pas versement).
- 0.5 pt une et une seule boucle `while`.

- 0.5 pt modification de la valeur de la dette dans la boucle (0.25 pt si mauvaise modification)
- 0.25 pt déclaration et initialisation du compteur d'années (une erreur de 1 est acceptée)
- 0.25 pt incrément du compteur d'année à chaque tour de boucle
- 0.25 pt affichage dans la boucle
- 0.25 pt affichage des bonnes valeurs avec les bonnes conversions

Question D. Le dernier versement est trop important. Comment modifier votre programme de manière à ne rembourser que le reste à payer, dernière année ? La dernière ligne affichée sera alors :

 1 pt
9 min

annee 5, interets : 1.01294, versement : 17.8952, dette : 0


Correction. juste avant affichage dans le while on insère :

```
if (dette < 0)
{
    versement = versement + dette; /* on supprime le trop versé */
    dette = 0;
}
```

Barème. Si tout juste : 1 pt sinon maxi 0.75 pt.

- 0.25 pt pour un if dans le while (ou après si condition du while modifiée)
- 0.25 pt pour un pour dette mise à zéro avant affichage
- 0.25 pt pour un versement égal au reste à percevoir avant affichage

Question E. Si le versement est inférieur aux intérêts, que dire de l'exécution de ce programme ?

 1 pt
9 min

Correction. Si le versement est inférieur aux intérêts la première année, la dette va augmenter et elle ne sera jamais nulle (ou négative). Donc le programme va boucler et la dette atteindre une valeur représentant l'infinie.

Barème. 0.5 pt si mention du fait que le programme ne s'arrête jamais.

2.3 Faut-il renoncer à rembourser la dette publique ?

Un arbre de décision est un graphe particulier où les nœuds sont des questions et les arêtes sont les réponses à ces questions. Il se lit de haut en bas. On avance dans l'arbre en répondant aux questions. Les nœuds les plus bas jouent le rôle particulier de classes de réponse au problème initial.

Ici, il y a deux classes de réponse : « Ne plus rembourser » et « Ne rien changer ». Par exemple, si les marchés sont inquiets, si la dette est strictement supérieure à 0.5 fois le PIB et que la population refuse l'austérité, alors on ne rembourse plus.

Vous utiliserez quatre variables dont vous coderez les valeurs avec des constantes symboliques. Les trois premières représentent les propriétés du jour courant pour prendre la décision, la dernière représente la décision à prendre :

- **marches** est un entier représentant l'état des marchés, elle peut valoir un entier signifiant PANIQUE, ou un entier signifiant INQUIETUDE ou un entier signifiant CONFIANCE.
- **dette** est un nombre à virgule représentant la dette en proportion du PIB (une valeur de 0.5 représente 50% du PIB).
- **refus** est un entier représentant le refus de l'austérité qui peut avoir la valeur usuelle TRUE (si la population refuse l'austérité) ou la valeur usuelle FALSE (sinon).
- **rembourser** est un entier représentant la décision à prendre et peut avoir la valeur TRUE (s'il faut continuer de rembourser) ou FALSE (s'il faut arrêter de rembourser). Sa valeur initiale est TRUE.

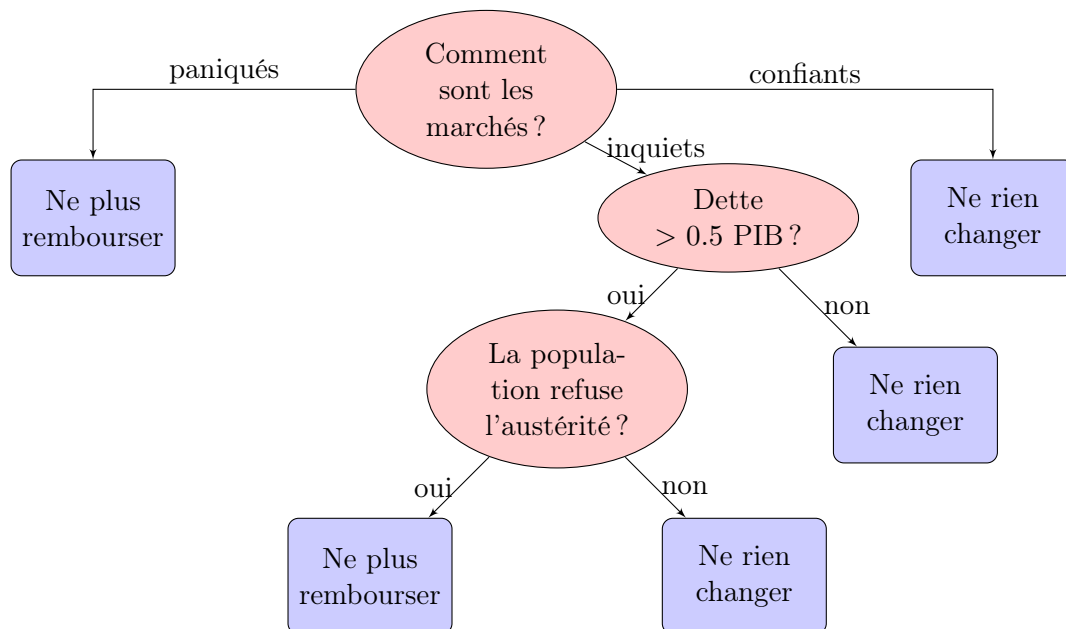


FIGURE 2 – Décider s’il faut continuer de rembourser la dette

Votre programme sera en deux parties. Une première partie concernera la prise de décision sans affichage, en fonction des valeurs de `marchés`, `dette`, `refus`. Vous donnerez des valeurs de votre choix à ces trois variables mais le programme doit fonctionner pour tous les autres choix possibles. À la fin de cette première partie `rembourser` contiendra la valeur correcte pour la décision prise (TRUE ou FALSE). La seconde partie exploitera la valeur de `rembourser` pour effectuer l’affichage.

Question F. Écrire le programme complet en distinguant bien les deux parties.

4 pt
36 min

Correction.

Barème. 4 pt si tout juste. Sinon on fait un maximum de 3.5 pt en sommant :

- 0.5 pt pour le codage `TRUE`, `FALSE`, `PANIQUE`, `INQUIETUDE`, `CONFIANCE` à l’aide de `define` s’il est correct (pas d’erreur de syntaxe, comme un signe égal), retirer une pénalité de 0.25 s’il a trop de constantes symboliques. S’il y a par exemple une constante symbolique pour définir la valeur d’initialisation de la distance.
- 0, 5 pt pour des déclarations avec initialisations correctes utilisant les constantes symboliques (0.25 pt si sans les constantes symboliques (par exemple 1 au lieu de `TRUE`)).
- 0, 5 pt pour une indentation correcte (si il y a matière à indenter!).
- Une pénalité de -0, 25 pt par oubli du `stdlib`, du `stdio` ou du `return final` (on ne sanctionne pas l’utilisation d’un 0 à la place du `EXIT_SUCCESS`).
- Pour l’arbre de décision : on part d’un total de 2.5 pt et on compte 0.5 pt point de pénalité par branche de l’arbre en erreur. S’il y a des boucles c’est zéro!
- 0.5 pt pour test sur `rembourser` et affichage correct.

3 Complexes en notation algébrique

Question G. Compléter le programme de la figure 4 :

5 pt
45 min

1. définir le type utilisateur `struct complexe_s`, pour stocker un nombre complexe en notation algébrique (sa partie réelle et sa partie imaginaire sous forme de nombres à virgule).
2. Dans le `main`, initialiser la valeur de la variable `z` à $-0.5 + 0.5i$ (il faut traduire en instructions C).

```

1  /* Declaration de fonctionnalites supplementaires */
2  #include <stdlib.h> /* pour EXIT_SUCCESS */
3  #include <stdio.h> /* pour printf() */
4
5  /* Declaration des constantes et types utilisateur */
6  #define TRUE 1
7  #define FALSE 0
8  #define PANIQUE 2
9  #define INQUIETUDE 1
10 #define CONFIANCE 0
11
12 /* Fonction principale */
13 int main()
14 {
15     /* Declaration et initialisation des variables */
16     int marches = PANIQUE;
17     double dette = 0.5;
18     int refus = TRUE;
19     int rembourser = TRUE;
20
21     /* Il y a trois possibilites exclusives les unes des autres, on ne note que les */
22     if (marches == PANIQUE) /* 1) accroche toi a l'euribor je retire le zinzin */
23     {
24         rembourser = FALSE;
25     }
26     if (marches == INQUIETUDE) /* 2) c'est grave docteur ? */
27     {
28         if (dette > 0.5) /* C'est rien ça va passer */
29         {
30             if (refus)
31             {
32                 rembourser = FALSE;
33             }
34         }
35     }
36
37     if (rembourser)
38     {
39         printf("Ne rien changer\n");
40     }
41     else
42     {
43         printf("Ne plus rembourser\n");
44     }
45
46     /* Valeur fonction */
47     return EXIT_SUCCESS;
48 }
49
50 /* Definition des fonctions utilisateur */

```

FIGURE 3 – Faut-il annuler la dette ? – Corrigé.

```

1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* déclaration constantes et types utilisateurs */
5  ...
6
7  /* déclaration de fonctions utilisateurs */
8  struct complexe_s multiplier(struct complexe_s x, struct complexe_s y);
9  ...
10
11 /* Fonction principale */
12 int main()
13 {
14     struct complexe_s z = ... ; /* à compléter pour traduire  $z = -0.5 + 0.5i$  */
15     struct complexe_s resultat;
16
17     /* calcul de  $(z * z) + z$  */
18     ....
19
20     /* affichage du résultat */
21     afficher(resultat);
22
23     /* valeur fonction */
24     return EXIT_SUCCESS;
25 }
26
27 /* définition de fonctions utilisateurs */
28 ...

```

FIGURE 4 – Calculs dans les complexes

3. Dans le main, calculer $z = z^2 + z$ en faisant appel à des fonctions élémentaires sur les complexes, telles que `multiplier` (rappel : les opérations `+` et `*` du C ne sont pas définies sur des struct). Vous aurez besoin d'une autre fonction que `multiplier` pour effectuer le calcul, il faut la déclarer (lui donner un nom explicite).
4. Déclarer également la procédure `afficher` qui sert à afficher le résultat.
5. Définir au choix deux des trois fonctions utilisateur de ce programme.

Votre programme doit fonctionner et calculer des valeurs correctes pour n'importe quelle autre initialisation de la variable `z`. Vous pouvez répondre à ces cinq questions en même temps, en donnant le code complet de votre programme, ou signaler pour chaque question à quelles lignes du programme de la figure 4 vous insérez de nouvelles instructions.

Correction. Correction figure 5.

Barème. Si tout juste : 5 pt.

- 1 pt déclaration correcte syntaxiquement d'une structure contenant exactement deux champs double ou float, (même si le nom n'est pas le bon).
- 0.5 pt initialisation correcte de la variable `z`.
- 1.5 pt calcul exact sinon maxi 1 pt
 - 0.5 pt si au moins un appel syntaxiquement correct à `multiplier`.
 - 0.25 pt si il est fait mention d'une fonction d'addition (quel que soit le nom) prenant deux complexes et retournant deux complexes.
 - 0.5 pt déclaration correcte d'une fonction d'addition avec même prototype que la multiplication.
- 0.5 pt Déclaration correcte de `afficher`
- 1.5 pt si deux définitions justes, sinon maxi 1 pt avec 0.75 pt par fonction (0.5 pt si valeur de retour non déclarée). On ne récompense pas de troisième fonction !


```

1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* déclaration constantes et types utilisateurs */
5  struct complexe_s
6  {
7      double re;
8      double im;
9  };
10
11 /* déclaration de fonctions utilisateurs */
12 struct complexe_s multiplier(struct complexe_s x, struct complexe_s y);
13 struct complexe_s additionner(struct complexe_s x, struct complexe_s y);
14 void afficher(struct complexe_s x);
15
16 /* Fonction principale */
17 int main()
18 {
19     struct complexe_s z = {-0.5, 0.5};
20     struct complexe_s resultat;
21
22     /* calcul de (z * z) + z */
23     resultat = multiplier(z, z);
24     resultat = additionner(resultat, z);
25
26     /* affichage du résultat */
27     afficher(resultat);
28
29     /* valeur fonction */
30     return EXIT_SUCCESS;
31 }
32
33 /* définition de fonctions utilisateurs */
34 struct complexe_s multiplier(struct complexe_s x, struct complexe_s y)
35 {
36     struct complexe_s res;
37     res.re = x.re * y.re - x.im * y.im;
38     res.im = x.re * y.im + x.im * y.re;
39     return res;
40 }
41 struct complexe_s additionner(struct complexe_s x, struct complexe_s y)
42 {
43     struct complexe_s res;
44     res.re = x.re + y.re;
45     res.im = x.im + y.im;
46     return res;
47 }
48 void afficher(struct complexe_s x)
49 {
50     printf("%lg + %lg i\n", x.re, x.im); /*sans cas particuliers */
51 }


```

FIGURE 5 – Calculs dans les complexes – correction


4 Autres déclarations de fonctions

1. Déclarer (sans la définir) une procédure `menu` qui n'a pas d'entrée et affiche :

```
*****  
* MENU *  
*****
```

 0.5 pt
4 min

2. Déclarer (sans la définir) une fonction qui calcule la valeur absolue d'un réel.

 0.5 pt
4 min

Correction.

```
void menu();  
double valeur_absolue(double x);
```

Barème. Compter simplement 0.5 pt par déclaration correcte.