Université Paris-Nord — L1
Institut Galilée — Année 2011-2012
Licence 1 - 1er semestre

Éléments d'informatique : partiel de mi-semestre

Durée : 3 heures.

Documents autorisés : Aucun.

Recommandations : Un barème vous est donné à titre indicatif afin de vous permettre de gérer votre temps. Vous pouvez traiter les questions dans l'ordre de votre choix, en faisant bien apparaître la numérotation alphabétique. Les premières questions sont plus faciles que les dernières. Il est recommandé de faire tourner à la main sur un exemple vos programmes pour s'assurer qu'il sont corrects.

1 Étude de programmes et questions de cours (*8 points*)

1.1 Questions de cours

Question A. Rappeler les cinq étapes de la compilation.

1 pt
9 min

Correction. Les cinq étapes de la compilation sont :
- l'analyse lexicale,
- l'analyse syntaxique,
- l'analyse sémantique,
- la génération du code,
- l'édition de liens.

Barème. *Si tout juste* $\boxed{1\,\text{pt}}$*, sinon :*
- *S'il ne manque que génération du code* $\boxed{0.75\,\text{pt}}$*, sinon :*
- $\boxed{0.5\,\text{pt}}$ *s'il manque une ou deux étapes ou s'il n'en manque aucune mais qu'elles sont dans le désordre (quel que soit le niveau de désordre). Sinon :*
- $\boxed{0.25\,\text{pt}}$ *si il y a au moins dans cet ordre : analyse syntaxique et analyse sémantique.*

Question B. Expliquer le fonctionnement de l'instruction `#define`. Vous pouvez prendre comme exemple le programme de la figure 1.

1 pt
9 min

Correction. L'instruction `#define` s'adresse au préprocesseur du compilateur, elle instruit le préprocesseur de l'existence d'un nom (par convention noté en majuscule) dans la suite du texte du programme et d'une expression qui devra se substituer à ce nom avant de commencer la compilation. Ce nom est appelé constante symbolique, il est en général plus clair pour le programmeur que l'expression qui le remplacera et facilite ainsi une relecture non ambiguë du programme (deux constantes symboliques définiront des noms différents pour des rôles différents, même si les expressions de substitution sont identiques).

Dans le programme de la figure 1, ligne 5, l'instruction `#define` définit une constante symbolique de nom `TAILLE` qui sera substituée par l'expression `3`. Cette constante est utilisée pour référer à la taille du tableau `t` dans la suite du programme.

Barème. 1 pt *si constante symbolique, préprocesseur, TAILLE remplacé par 3.* 0.5 pt *si seulement notion de constantes et pas de mise en relation avec la substitution au moment de la compilation.*

```
1   #include <stdlib.h> /* EXIT_SUCCESS */
2   #include <stdio.h> /* printf, scanf */
3
4   /* déclaration constantes et types utilisateurs */
5   #define TAILLE 3
6
7   /* Fonction principale */
8   int main()
9   {
10    int t[TAILLE] = {12, 15, 10}; /* tableau */
11
12
13
14    /* calcul du maximum (À FAIRE) */
15
16
17
18
19
20    /* affichage du résultat */
21    printf("Le maximum est %d\n", maximum);
22
23    /* valeur fonction */
24    return EXIT_SUCCESS;
25  }
```

FIGURE 1 – Calcul du maximum d'un tableau (à compléter)

1.2 Maximum des éléments d'un tableau

Nous voulons écrire un programme qui calcule le maximum des éléments d'un tableau d'entiers positifs. Une partie du programme est déjà écrite, figure 1, et Pippo pense qu'il ne reste plus qu'à écrire la partie qui calcule effectivement le maximum.

Question C. Avant de continuer Pippo veut tester son programme, mais la compilation échoue. Expliquer pourquoi, quelle étape de la compilation échoue précisément et ce qu'il manque pour que la compilation réussisse.

1 pt
9 min

Correction. La compilation échoue car la variable `maximum` est utilisée sans avoir été déclarée. C'est à l'analyse sémantique que cette erreur est détectée puisque c'est l'étape où les objets manipulés par le programme et les actions sur ces objets sont analysés.

Barème. 1 pt *dont* 0.5 pt *pour* déclaration de la variable `maximum` *plus* 0.5 pt *pour* analyse sémantique *ou* 0.25 pt *si* avant la génération du code objet *et/ou* avant l'édition de lien.

Question D. Compléter le programme pour qu'il calcule effectivement le maximum des éléments du tableau (pour n'importe quel tableau contenant des entiers positifs et de taille `TAILLE` fixée arbitrairement).

2 pt
18 min

Exemple. Pour le tableau donné dans le code, une fois terminé, le programme affichera :

```
Le maximum est 15
```

Correction. Voir figure 2.

```
11      int maximum = 0; /* maximum initial: plus petite valeur possible */
12      int i; /* variable de boucle */
13
14      /* calcul du maximum */
15      for (i = 0; i < TAILLE; i = i + 1) /* pour chaque element de t */
16      {
17          if (maximum < t[i]) /* t[i] > max(t[0], ... , t[i - 1]) */
18          {
19              maximum = t[i]; /* t[i] = max(t[0], ..., t[i]) */
20          }
21      }
22
```

FIGURE 2 – Complément de programme

Barème. $\boxed{2.5\,\text{pt}}$ *(barème augmenté) pour le code juste (compilation correction quel que soit TAILLE). Sinon, maxi* $\boxed{2\,\text{pt}}$ *:*

- *pas de boucle on met* $\boxed{0\,\text{pt}}$ *quelque soit le contenu.*
- *pas de points en moins ou en plus pour la déclaration de la variable de boucle (redite sous une autre forme de la première question) ou pour l'initialisation de maximum à zéro ou à la première case du tableau.*
- $\boxed{0.5\,\text{pt}}$ *une (et une seule boucle (for, while accepté)*
- $\boxed{0.5\,\text{pt}}$ *la boucle parcours effectivement un tableau de taille TAILLE (TAILLE - 1 admis si l'initialisation du maximum est faite à* `t[0]`*).*
- $\boxed{0.5\,\text{pt}}$ *pour un if imbriqué dans la boucle.*
- $\boxed{0.5\,\text{pt}}$ *si le if est correct, c'est à dire s'il réalise le bon test (comparaison avec le maximum courant) et change la valeur du maximum courant en conséquent.*

1.3 Trace d'un programme

3 pt
27 min

Question E. Simuler l'exécution du programme figure 3, en réalisant sa **trace**.

Rappel. La trace représente l'exécution du programme par un tableau à $n + 2$ colonnes : la première colonne contient le numéro de la ligne exécutée ; les n colonnes suivantes, les variables du programme (n à déterminer) et la dernière colonne, l'affichage réalisé par le programme à l'écran. La trace commence par une ligne nommant les colonnes, puis la ligne reportant les valeurs initiales des variables, après quoi seules les lignes modifiant l'état mémoire du programme ou l'affichage sont à reporter dans le tableau. L'utilisateur doit saisir trois entiers : considerer, comme indiqué en commentaires, qu'il saisit 1 puis 2 puis 3.

Correction. Voir table 1.

```c
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */

/* declarations de constantes et types utilisateurs */

/* declarations de fonctions utilisateurs */

/* fonction principale */
int main()
{
    int terme;
    int raison;
    int n;
    int i;
    int serie = 0;

    printf("Entrez le premier terme, la raison puis le nombre de termes : ");
    scanf("%d",&terme);  /* 1 est saisi par l'utilisateur */
    scanf("%d",&raison); /* 2 est saisi par l'utilisateur */
    scanf("%d",&n);      /* 3 est saisi par l'utilisateur */

    for (i = 0; i < n; i = i + 1)
    {
        serie = serie + terme;
        terme = terme * raison;
    }

    printf("somme des %d premiers termes de la suite : %d\n", n, serie);

    return EXIT_SUCCESS;
}
```

FIGURE 3 – Faire la trace du programme

`main()` (l'utilisateur saisit 1, puis 2, puis 3)

numéro de ligne	`terme`	`raison`	`n`	`i`	`serie`	**Affichage (sortie écran)**
initialisation	?	?	?	?	0	
17						Entrez le premier terme, la raison, puis le nombre de termes :
18	1					
19		2				
20			3			
22				0		
24					1	
25	2					
26				1		
24					3	
25	4					
26				2		
24					7	
25	8					
26				3		
28						somme des 3 premiers termes de la suite : 7
30	retourne `EXIT_SUCCESS`					

TABLE 1 – Trace du programme de la figure 3

Barème. *Compter :*

- $\boxed{0.5\,\text{pt}}$ *pour une ligne d'en-tête juste, autrement dit pour les colonnes (0 sinon)*
- $\boxed{0.5\,\text{pt}}$ *pour une ligne d'initialisation juste (0 sinon)*
- *aucune pénalité si la dernière ligne est absente (renvoie ....).*
- $\boxed{2\,\text{pt}}$ *pour le corps du main juste :*
  - $\boxed{-0,5\,\text{pt}}$ *de pénalités si des erreurs dans la partie saisie*
  - $\boxed{-0,5\,\text{pt}}$ *de pénalités si des erreurs dans l'un ou l'autre ou les deux affichages.*
  - $\boxed{-1\,\text{pt}}$ *de pénalité sur le for : si la boucle ne passe pas exactement par trois étapes avec i = 0, 1, 2 ou bien si la dernière affectation de* `i` *(à 3) n'est pas faite.*
  - *Pour toute autre ligne erronée (ligne 24 ou 25 ne compter qu'une seule fois) compter* $\boxed{-0.5\,\text{pt}}$ *de pénalité.*
- *Si des valeurs de variables non modifiées apparaissent sur certaines lignes compter également une pénalité de* $\boxed{-0.5\,\text{pt}}$.

2 Jours d'automne (*4 points*)

Cette année votre premier semestre se déroule sur trois saisons : l'été qui se termine le 22 septembre, l'automne qui commence le 23 septembre et se termine le 21 décembre et l'hiver qui commence le 22 décembre.

Soient deux variables entières `jour` et `mois`, représentant une date entre le premier septembre (1/9) et le 31 décembre (31/12) et que vous initialiserez à des valeurs de votre choix, prises dans cet intervalle (votre programme doit fonctionner pour n'importe quel choix correct de date).

Question F. Écrire un programme qui :

1. définit trois constantes symboliques pour représenter par trois nombres différents les trois saisons : `ETE`, `AUTOMNE`, `HIVER`. — 0.5 pt, 4 min
2. affiche la date représentée par les variables `jour` et `mois` — 0.5 pt, 4 min
3. trouve la saison correspondant à cette date — 2 pt, 18 min
4. affiche :
   - `C'est encore l'été`, si la date est en été ;
   - `C'est l'automne`, si la date est en automne ;
   - `Déjà l'hiver`, si la date du jour est en hiver. — 1 pt, 9 min

Vous séparerez clairement le calcul de la saison et la partie affichage dans votre programme. Pour cela vous utiliserez une variable entière `saison` dont les valeurs possibles seront : `ETE`, `AUTOMNE`, `HIVER`.

Exemples d'exécutions :

```
Date : 20 10
c'est l'automne

Date : 5 9
C'est encore l'ete

Date :  23 12
Deja l'hiver
```

Correction. Voir figure 4.

Barème.

- $\boxed{0.5\,\text{pt}}$ *pour les trois define avec trois valeurs quelconques mais différentes.*
- $\boxed{0.5\,\text{pt}}$ *pour l'affichage jour/mois*
- $\boxed{3\,\text{pt}}$ *pour calcul et affichage. Si ce programme est trop compliqué (des boucles ou autre) c'est zéro à cette partie sans regarder plus loin. Sinon :*
  - $\boxed{1\,\text{pt}}$ *si séparation calcul / affichage et que l'affichage en fonction de la valeur de la variable saison est correct (a des variations de syntaxe pret).*
  - $\boxed{0.5\,\text{pt}}$ *si fonctionne pour mois 10 et 11 (jour quelconque)*
  - $\boxed{0.5\,\text{pt}}$ *par transition dans le mois correctes (en décembre et en septembre),* $\boxed{0.25\,\text{pt}}$ *si c'est à un jour pret.*
- $\boxed{-0.5\,\text{pt}}$ *s'il y a des variables non declarées, ou si il n'y a pas de main.*
- *aucune pénalité pour le traitement des dates mals formatées ou en dehors de l'intervalle.*
- *pas de penalité si oublie d'un include ou du EXIT_SUCCESS.*

3 For ou while ?

3.1 Puissances successives d'un entier (*4 points*)

Question G. Écrire un programme qui demande un entier n à l'utilisateur et calcule puis affiche n^{10} (n puissance dix). 2 pt 18 min

Exemple d'exécution.

```
Donner un entier : 2
2 exposant 10 = 1024
```

Correction. Voir figure 5.

Barème.

- $\boxed{0.5\,\text{pt}}$ *pour la saisie meme si oublie du &*
- $\boxed{1.5\,\text{pt}}$ *pour le reste, zéro si pas de boucle sinon :*
  - *enlever* $\boxed{0.5\,\text{pt}}$ *si c'est un while*
  - *enlever* $\boxed{0.5\,\text{pt}}$ *par problème de déclaration, d'initialisation, ou d'include manquant.*
  - *enlever* $\boxed{0.5\,\text{pt}}$ *si erreur sur le nombre d'étapes de boucle.*
  - *on ne demande pas un affichage élaboré.*

Question H. Comment modifier simplement votre programme pour qu'il affiche le calcul des puissances successives de l'entier n saisi, de n^0 jusqu'à n^{10} ? 1 pt 9 min

Exemple d'exécution.

```
Donner un entier : 2
2 exposant 0 = 1
2 exposant 1 = 2
2 exposant 2 = 4
...
2 exposant 10 = 1024
```

```c
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */

/* 1) déclaration constantes et types utilisateurs */
#define ETE 1
#define AUTOMNE 2
#define HIVER 3

/* Fonction principale */
int main()
{
    int jour = 21;
    int mois = 9;
    int saison = AUTOMNE;

    /* 2 affichage */
    printf("Date : %d %d\n", jour, mois);


    /*3 calcul de la saison */
    if ((9 == mois) && (jour < 23))
    {
        saison = ETE;
    }
    if ((12 == mois) && (jour > 21))
    {
        saison = HIVER;
    }

    /* 3' optionnel (non demandé) */
    if ((mois < 9) || (mois > 12))
    {
      saison = 0;
      printf("Il n'y a plus de saisons !\n");
    }

    /*4 affichage */
    if (ETE == saison)
    {
        printf("C'est encore l'ete\n");
    }
    if (AUTOMNE == saison)
    {
        printf("C'est l'automne\n");
    }
    if (HIVER == saison)
    {
        printf("Deja l'hiver\n");
    }


   /* valeur fonction */
    return EXIT_SUCCESS;
}
```

FIGURE 4 – Jours d'automne

```c
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */
/* declarations de constantes et types utilisateurs */

/* declarations de fonctions utilisateurs */

/* fonction principale */
int main()
{
    int n = 0; /* base */
    int produit = 1;
    int i; /* var de boucle */

    /* saisie */
    printf("Donner un entier : ");
    scanf("%d", &n);

    /* calcul */
    for (i = 0; i < 10; i = i + 1) /* faire 10 fois
                                      la multiplication par n */
    {
        produit = produit * n;
    }

    /* affichage */
    printf("%d exposant %d = %d\n", n, 10, produit);

    return EXIT_SUCCESS;
}
```

FIGURE 5 – Puissance dix

Correction. Il suffit de placer l'affichage dans la boucle, au début du bloc.

```
/* calcul et affichage */
for (i = 0; i < 10; i = i + 1) /* faire 10 fois
                                  la multiplication par n */
{
    /* affichage */
    printf("%d exposant %d = %d\n", n, i, produit);
    produit = produit * n;
}
/* affichage */
printf("%d exposant %d = %d\n", n, 10, produit);
```

Barème. *Compter* $\boxed{0\,\text{pt}}$ *pour toute autre solution. Pénalité de* $\boxed{-0.5\,\text{pt}}$ *si n'affiche pas l'exposant ou pas la bonne valeur d'exposant. Ne pas enlever de point si l'affichage ne va que jusqu'à l'exposant 9.*

Question I. Comment modifier simplement votre programme pour qu'il affiche le calcul des puissances successives de l'entier n uniquement à partir de l'exposant 5 ?

1 pt
9 min

Exemple d'exécution.

```
Donner un entier : 2
2 exposant 5 = 32
2 exposant 6 = 64
...
2 exposant 10 = 1024
```

Correction. il suffit d'encapsuler l'affichage de l'intérieur de boucle dans un test :

```
/* calcul et affichage */
for (i = 0; i < 10; i = i + 1) /* faire 10 fois
                                  la multiplication par n */
{
    /* affichage */
    if (i > 4)
    {
        printf("%d exposant %d = %d\n", n, i, produit);
    }
    produit = produit * n;
}
/* affichage */
printf("%d exposant %d = %d\n", n, 10, produit);
```

Barème. *Autre solution admise : deux boucles (i allant de 0 à 4 puis i allant de 5 à 9) à la suite l'une de l'autre. Sinon zéro.*

3.2 Unicité des éléments d'un tableau (*4 points*)

Nous disposons d'un tableau t de N entiers. Utiliser une constante symbolique pour N. Nous souhaitons savoir si chaque entier apparaissant dans le tableau n'y apparaît qu'une seule fois, autrement dit on veut savoir si chaque entier est unique dans le tableau.

Question J. Écrire un programme qui, étant donné un tableau initalisé t, teste si le premier élément du tableau est unique et affiche `Vrai` si c'est le cas, `Faux` sinon.

2 pt
18 min

Correction. Voir figure 6.

```
1   #include <stdlib.h> /* EXIT_SUCCESS */
2   #include <stdio.h> /* printf, scanf */
3
4   /* déclaration constantes et types utilisateurs */
5   #define N 3
6   #define TRUE 1
7   #define FALSE 0
8
9   /* Fonction principale */
10  int main()
11  {
12    int t[N] = {10,10,15};
13    int unique = TRUE;
14    int i; /* var de boucle */
15
16    i = 1;
17    while (unique && (i < N)) /* tant que t[0] est unique */
18    {
19      if (t[i] == t[0]) /* t[0] n'est pas unique */
20      {
21        unique = FALSE;
22      }
23      i = i + 1; /* case suivante */
24    }
25
26    /* affichage du résultat */
27    if (unique)
28    {
29      printf("Vrai\n");
30    }
31    else
32    {
33      printf("Faux\n");
34    }
35    /* valeur fonction */
36    return EXIT_SUCCESS;
37  }
```

FIGURE 6 – Unicité des éléments d'un tableau

Barème. $\boxed{2\,\text{pt}}$ *si arrive au résultat avec une seule boucle (for ou while). L'affichage doit être correct mais on accepte les variantes : "machin n'est pas unique / "machin est unique" etc.*

Sinon maximum $\boxed{1,5\,\text{pt}}$ *:* $\boxed{0,5\,\text{pt}}$ *s'il y a une seule boucle ;* $\boxed{0,5\,\text{pt}}$ *de plus si c'est un while ; ajouter* $\boxed{0,5\,\text{pt}}$ *si utilise une variable booléenne (déclarée et initialisée, et les define TRUE et FALSE).*

Question K. Écrire un programme qui étant donné un tableau initialisé t, teste si tous les éléments sont uniques et affiche `Vrai` si c'est le cas, `Faux` sinon.

2 pt
18 min

Correction. Voir figure 7.

Barème. $\boxed{2\,\text{pt}}$ *si arrive au résultat avec deux boucles (avec affichage mais variantes acceptée).*

Sinon maximum $\boxed{1,5\,\text{pt}}$ *:* $\boxed{0,5\,\text{pt}}$ *s'il y a deux boucles avec variables de boucles différentes (quelles que soient les boucles et les bornes) ;* $\boxed{0,5\,\text{pt}}$ *si elles sont toutes deux des while ;* $\boxed{0,5\,\text{pt}}$ *s'il a un algorithme correct en français en commentaire ou dans le texte ;* $\boxed{0,5\,\text{pt}}$ *si utilise une variable boolénnee ou deux variables booléennes.*

Question bonus (*2 points*)

Cette question est difficile et elle sera notée de manière très stricte. Il vaut beaucoup mieux traiter toutes les autres questions avant de chercher à y répondre.

Question L. Écrire un programme qui :

1. demande à l'utilisateur d'entrer un nombre n dont on supposera qu'il est strictement supérieur à 1.
2. tant que n est différent de 1, répète l'étape suivante :

   calculer une nouvelle valeur de n ainsi :
   – si n est pair, le diviser par deux
   – si n est impair, le multiplier par trois et lui ajouter un
   et afficher la nouvelle valeur de n.
3. affiche le nombre de fois où l'étape précédente a été répétée, et la plus grande valeur prise par n au cours de ce calcul.

Ce programme termine-t-il toujours ?

Correction. Cette question bonus sera traitée en TD à la fin du semestre.

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf, scanf */
3
4  /* déclaration constantes et types utilisateurs */
5  #define N 4
6  #define TRUE 1
7  #define FALSE 0
8
9  /* Fonction principale */
10 int main()
11 {
12   int t[N] = {10,12,15,12};
13   int unique = TRUE;
14   int i; /* var de boucle */
15   int j; /* var de boucle */
16
17   j = 0; /* premiere case */
18   while (unique && (j < N)) /* pour chaque case, verifier que la case
19   est unique */
20   {
21     i = j + 1; /* case suivant t[j] */
22     while (unique && (i < N)) /* tant que t[j] est unique */
23     {
24       if (t[i] == t[j]) /* t[j] n'est pas unique */
25       {
26         unique = FALSE;
27       }
28       i = i + 1; /* case suivante */
29     }
30     j = j + 1; /* case suivante */
31   }
32   /* affichage du résultat */
33   if (unique)
34   {
35     printf("Vrai\n");
36   }
37   else
38   {
39     printf("Faux\n");
40   }
41   /* valeur fonction */
42   return EXIT_SUCCESS;
43 }
```

FIGURE 7 – Unicité de tous les éléments du tableau