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

Travaux dirigés 12

Correction. Note aux chargés de TD : ce sont des exercices typiques qu'ils auront à l'examen : ils doivent savoir écrire des fonctions et les programmes qui les utilisent.

La deuxième partie est tirée de l'examen de l'année dernière. La fin du TD doit être utilisée pour les révisions demandées par les étudiants. **Attention :** Certaines parties sont sans correction.

1 Écriture et appel de fonctions

1.1 Un programme de géométrie

Nous souhaitons écrire un programme de géométrie permettant le calcul d'aires de formes simples : l'utilisateur spécifie les dimensions des figures et le programme calcule les aires de ces figures. Afin de tester les fonctions, un premier programme est écrit :

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

/* Declaration des constantes et types utilisateurs */
#define PI 3.1415926

/* Declaration des fonctions utilisateurs */
/* calcule l'aire d'un rectangle */
double aire_rectangle(double longueur,double largeur);
/* calcule l'aire d'un cercle */
double aire_cercle(double rayon);
/* calcule l'aire d'un triangle */
double aire_triangle(double base,double hauteur);

int main()
{
    double longueur;
    double largeur;
    double cote;
    double rayon;
    double base;
    double hauteur;

    printf("Entrez la longueur et largeur du rectangle : ");
    scanf("%lg",&longueur);
    scanf("%lg",&largeur);
    printf("L'aire du rectangle est %g.\n",aire_rectangle(longueur,largeur));

    printf("Entrez le rayon du cercle : ");
    scanf("%lg",&rayon);
    printf("L'aire du cercle est %g.\n",aire_cercle(rayon));

    printf("Entrez la base et la hauteur du triangle : ");
    scanf("%lg",&base);
    scanf("%lg",&hauteur);
    printf("L'aire du triangle est %g.\n",aire_triangle(base,hauteur));
```

```
    return EXIT_SUCCESS;
}

/* Definitions des fonctions utilisateurs */
```

– Est-ce que le programme compile ? Est-ce que l'édition de liens réussit ? **Les réponses doivent être succintement expliquées**.

Correction. La compilation réussit si l'analyse lexicale, syntaxique et sémantique réussissent. Le programme compile parce qu'il n'y a pas d'erreurs lexicales, syntaxiques et que l'utilisation des variables et des fonctions est conforme à leur déclaration. Par contre l'édition de liens échoue car le code des fonctions est manquant pour produire un exécuatble. NB : ce n'est pas le cas de printf et scanf qui ont leur code disponible sur le disque dur dans un répertoire connu de l'éditeur de liens.

– Définir les fonctions manquantes.

Correction.

```
1   /* Declaration de fonctionnalites supplementaires */
2   #include <stdlib.h> /* EXIT_SUCCESS */
3   #include <stdio.h> /* printf, scanf */
4
5   /* Declaration des constantes et types utilisateurs */
6   #define PI 3.1415926
7
8   /* Declaration des fonctions utilisateurs */
9   /* calcule l'aire d'un rectangle */
10  double aire_rectangle(double longueur,double largeur);
11  /* calcule l'aire d'un cercle */
12  double aire_cercle(double rayon);
13  /* calcule l'aire d'un triangle */
14  double aire_triangle(double base,double hauteur);
15
16  int main()
17  {
18      double longueur;
19      double largeur;
20      double cote;
21      double rayon;
22      double base;
23      double hauteur;
24
25      printf("Entrez la longueur et largeur du rectangle : ");
26      scanf("%lg",&longueur);
27      scanf("%lg",&largeur);
28      printf("L'aire du rectangle est %g.\n",aire_rectangle(longueur,largeur));
29
30      printf("Entrez le rayon du cercle : ");
31      scanf("%lg",&rayon);
32      printf("L'aire du cercle est %g.\n",aire_cercle(rayon));
33
34      printf("Entrez la base et la hauteur du triangle : ");
35      scanf("%lg",&base);
36      scanf("%lg",&hauteur);
37      printf("L'aire du triangle est %g.\n",aire_triangle(base,hauteur));
38
39      return EXIT_SUCCESS;
40  }
41
42  /* Definitions des fonctions utilisateurs */
43  /* calcule l'aire d'un rectangle */
```

```
44  double aire_rectangle(double longueur,double largeur)
45  {
46      return longueur * largeur;
47  }
48
49  /* calcule l'aire d'un cercle */
50  double aire_cercle(double rayon)
51  {
52      return PI * rayon * rayon;
53  }
54
55  /* calcule l'aire d'un triangle */
56  double aire_triangle(double base,double hauteur)
57  {
58      return (base * hauteur) / 2;
59  }
60
```

– Afin de s'assurer de la correction du programme, faire sa trace.

1.2 Calcul de la constante e

Correction. L'essentiel ici est la réutilisation de la fonction factorielle (déjà vue) et de trouver un programme le plus simple possible pour tester, sachant que $e = 2,718281828459045235360287 4....$ Vous pouvez expliquer que factorielle(n) va très rapidement faire un dépassement de capacité et le calcul ne sera plus valide.

– Écrire la fonction `factorielle` qui prend en entrée un entier positif ou nul n et qui renvoie sa factorielle $n!$, avec la convention $0! = 1$.
– Écrire la fonction `e_approchee` qui prend en entrée un entier n et qui renvoie $\sum_{i=0}^{i=n} \frac{1}{n!}$. Cette serie converge vers la constante e, la constante de Néper, la base des logarithmes naturels. On utilisera la fonction `factorielle`
– Écrire un complet programme simple qui permet de tester `e_approchee`.

Correction.

```
1   /* Declaration de fonctionnalites supplementaires */
2   #include <stdlib.h> /* EXIT_SUCCESS */
3   #include <stdio.h> /* printf, scanf */
4
5   /* Declaration des fonctions utilisateurs */
6   /* calcule de factorielle */
7   int factorielle(int n);
8   /* calcule de la constante e approchee */
9   double e_approchee(int n);
10
11  int main()
12  {
13      int n;
14
15      printf("Entrez l'indice de la série : ");
16      scanf("%d",&n);
17
18      printf("Une valeur approchee de la constante de Neper est : %g.\n",e_approchee(n));
19
20      return EXIT_SUCCESS;
21  }
22
23  /* Definitions des fonctions utilisateurs */
24  /* calcule de factorielle */
25  int factorielle(int n)
26  {
```

```
27      int i; /* Var. de boucle */
28      int res = 1; /* resultat */
29
30      for (i = 1; i <= n; i = i + 1) /* Pour i = 1, 2, ..., n */
31      {
32          res = res * i; /* mettre i dans le produit */
33      }
34
35      /* Valeur fonction */
36      return res;
37  }
38
39  /* calcule de la constante e approchee */
40  double e_approchee(int n)
41  {
42      int i; /* Var. de boucle */
43      double res = 0; /* elt neutre de l'addition */
44
45      for (i = 0; i <= n; i = i + 1) /* Pour i = 1, 2, ..., n */
46      {
47          res = res + 1.0 / factorielle(i); /* attention on ne veut pas de division entiere */
48      }
49
50      /* Valeur fonction */
51      return res;
52  }
```

2 Révision

2.1 Calcul du quotient d'une division entière par soustractions successives (*4 points*)

Nous voulons écrire une fonction `quotient_par_soustraction` qui prend deux entiers strictement positifs a et b en paramètre, et retourne le quotient de la division entière de a par b. Le nombre a est appelé le dividende et b le diviseur. Cette fonction doit calculer le quotient par soustractions successives **sans utiliser l'opérateur / du C (division entière)**. Le programme principal est déjà écrit pour pouvoir tester la fonction :

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

/* declarations constantes et types utilisateurs */

/* declarations de fonctions utilisateurs */

int main()
{
  int dividende;
  int diviseur;

  printf("Entrez le dividende et le diviseur : ");
  scanf("%d",&dividende);
  scanf("%d",&diviseur);

  while(dividende <= 0 || diviseur <= 0) /* pas les deux strictement positifs */
  {
      printf("Erreur. Entrez le dividende et le diviseur : ");
      scanf("%d",&dividende);
      scanf("%d",&diviseur);
  }
```

```
  printf("le quotient de la division euclidienne est : %d\n",
         quotient_par_soustraction(dividende,diviseur));

  return EXIT_SUCCESS;
}

/* definitions de fonctions utilisateurs */
```

1. Est-ce que le programme compile ?

   Correction. La compilation réussit si l'analyse lexicale, syntaxique et sémantique réussissent. Le programme ne compile pas parce que l'analyse sémantique échoue : une fonction est appelée sans avoir été déclarée, on ne peut pas vérifié si le nom est correct, si les types d'entrée et sortie sont les bons.

2. Compléter le programme pour que l'édition de liens réussise et produise un exécutable.

   Correction. Il faut définir et déclarer la fonction `quotient_par_soustraction`.

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

/* declarations constantes et types utilisateurs */

/* declarations de fonctions utilisateurs */
int quotient_par_soustraction(int a,int b);

int main()
{
  int dividende;
  int diviseur;

  printf("Entrez le dividende et le diviseur : ");
  scanf("%d",&dividende);
  scanf("%d",&diviseur);

  while(dividende <= 0 || diviseur <= 0) /* pas les deux strictement positifs */
  {
      printf("Erreur. Entrez le dividende et le diviseur : ");
      scanf("%d",&dividende);
      scanf("%d",&diviseur);
  }

  printf("le quotient de la division euclidienne est : %d\n",
         quotient_par_soustraction(dividende,diviseur));

  return EXIT_SUCCESS;
}

/* definitions de fonctions utilisateurs */
/* a = qb + r. on peut enlever b à a q fois avant d'obtenir le reste r < b */
int quotient_par_soustraction(int a,int b)
{
  int quotient = 0; /* le quotient de a par b */
```

```
    while(a >= b)
    {
        a = a - b;
        quotient = quotient + 1;
    }

    return quotient;
}
```

Deux exemples d'exécution du programme sont :

```
Entrez le dividende et le diviseur : 23 4
le quotient de la division euclidienne est : 5
Entrez le dividende et le diviseur : 1 4
le quotient de la division euclidienne est : 0
```

2.2 Exercices sur des tableaux, sans fonctions (*5 points*)

Pour les deux exercices suivants, tout le traitement sera effectué dans le `main`, sans faire appel à des fonctions utilisateurs.

2.2.1 Somme de deux vecteurs (*1,5 points*)

Rappel : la somme de deux vecteurs du plan se fait terme à terme comme ceci :

$$(x, y) + (x', y') = (x + x', y + y').$$

Nous voulons faire un programme qui réalise la somme de deux vecteurs de dimension N. Un vecteur sera représenté par un tableau.

Écrire un programme qui, étant donnés deux tableaux de réels, u et v, de tailles N, calcule dans un tableau w de taille N la somme terme à terme de u et de v, puis affiche le contenu de w. Définir `N` comme une constante symbolique valant 2. Les tableaux u et v seront initialisés à des valeurs de votre choix.

Tout le traitement sera effectué dans le `main`*, sans faire appel à des fonctions utilisateurs.*

2.2.2 Unicité des éléments d'un tableau (*3,5 points*)

Nous disposons d'un tableau t de N caractères et nous souhaitons savoir si chaque caractère apparaissant dans le tableau n'y apparaît qu'une seule fois, autrement dit on veut savoir si chaque caractère est unique.

1. 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. 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.

Tout le traitement sera effectué dans le `main`*, sans faire appel à des fonctions utilisateurs.*