E.N.S.E.I.R.B. Année 2005-2006
Filière Électronique $1^{ère}$ année
Filière Informatique $1^{ère}$ année
Filière Télécom $1^{ère}$ année

PROGRAMMATION IMPÉRATIVE - LANGAGE C

CORRIGÉ

Feuille 1

Q1 . En C standard, les commentaires s'écrivent entre :

(A☐) `*/.../*`
(B☐) `*/.../*`
(C☐) `/*.../*`
(D☒) `/*.../*`

Q2 . Un bloc d'instructions entre accolades `{...}` :

(A☐) Doit obligatoirement être suivi d'un point-virgule
(B☐) Peut éventuellement être suivi d'un point-virgule
(C☐) Ne doit être suivi d'un point-virgule que si c'est une définition de fonction
(D☒) Ne doit jamais être suivi d'un point-virgule

Q3 . Un test s'écrit :

(A☐) `if (`*expression*`) ; then` *instruction*
(B☒) `if (`*expression*`)` *instruction*
(C☐) `if [` *expression* `] ; then` *instruction*
(D☐) `if (`*expression*`) then` *instruction*

Q4 . Pour mettre le contenu de la variable `b` dans la variable `a` si `a` est impair, et diviser `a` par 2 si `a` est pair, on peut écrire :

(A☐) `a = ((a % 2) = 0) ? /= 2 : b;`
(B☐) `a = ((a % 2) == 0) ? /= 2 : b;`
(C☒) `if (a % 2) a = b; else a /= 2;`
(D☐) `a = (a % 2) ? a / 2 : b;`

Q5 . Pour disposer dans une fonction `f` d'une variable `i` dont la valeur est préservée entre chaque appel et qui ne soit pas visible de l'extérieur de la fonction `f`, il faut la définir en tant que :

(A☒) « `static int i;` » à l'intérieur de `f`
(B☐) « `int i;` » à l'intérieur de `f`
(C☐) « `static int i;` » à l'extérieur de `f`
(D☐) « `int i;` » à l'extérieur de `f`

Q6 . Pour récupérer au clavier un entier et deux caractères à placer dans une variable `i` et les deux cases `t[0]` et `t[1]` d'un tableau de deux caractères, on peut écrire :

(A[X]) `scanf ("%d%c%c", &i, t, t + 1)`
(B[ ]) `scanf ("%d%c%c", &i, *t, *(t + 1))`
(C[ ]) `scanf ("%d%c%c", i, t[0], t[1])`
(D[ ]) `scanf ("%d%c%c", &i, t)`

Q7 . Le corps de la boucle « `for (i = 0; i <= 10; i ++)` *corps*`;` » s'exécute :

(A[ ]) 3 fois
(B[ ]) 7 fois
(C[ ]) 10 fois
(D[X]) 11 fois

Q8 . Le corps de la boucle « `for (i = 1; i < 10; ) { ++ i;` *corps*`; }` » s'exécute :

(A[ ]) 3 fois
(B[X]) 9 fois
(C[ ]) 11 fois
(D[ ]) Aucune des réponses précédentes

Q9 . Le corps de la boucle « `for (i = 0; i > 10; i ++)` *corps*`;` » s'exécute :

(A[X]) 0 fois
(B[ ]) 6 fois
(C[ ]) 10 fois
(D[ ]) Aucune des réponses précédentes

Q10 . Le corps de la boucle « `for (i = 1; i < 10; i ++)` *corps*`; i ++;` » s'exécute :

(A[ ]) 2 fois
(B[ ]) 5 fois
(C[ ]) 10 fois
(D[X]) Aucune des réponses précédentes

Q11 . Le corps de la boucle « `for (i = 0; i ++ < 10; i ++)` *corps*`;` » s'exécute :

(A[ ]) 1 fois
(B[X]) 5 fois
(C[ ]) 9 fois
(D[ ]) 10 fois

Q12 . Le corps de la boucle « `for (i = 0; ++ i < 10; )` *corps*`;` » s'exécute :

(A[ ]) 1 fois
(B[ ]) 4 fois
(C[X]) 9 fois
(D[ ]) 10 fois

Q13 . Le corps de la boucle « `for (i = 1; i < 10; )` *corps*`; i ++;` » s'exécute :

(A[ ]) 1 fois
(B[ ]) 6 fois
(C[ ]) 9 fois
(D[X]) Aucune des réponses précédentes

Q14 . Le corps de la boucle « `for (i = 1; i < 10; i ++) {` *corps*`; if (i > 4) break; }` » s'exécute :

(A[ ]) 2 fois
(B[ ]) 3 fois
(C[ ]) 4 fois
(D[X]) 5 fois

Q15 . Le corps de la boucle « `for (i = 1; i < 10; i ++) {` *corps*`; if (i > 4) continue; }` » s'exécute :

(A[ ]) 2 fois
(B[ ]) 5 fois
(C[X]) 9 fois
(D[ ]) Aucune des réponses précédentes

Q16 . Le corps de la boucle « `for (i = 1; i < 10; i ++) { if (i % 2) continue;` *corps*`; }` » s'exécute :

(A☐) 3 fois
(B☒) 4 fois
(C☐) 9 fois
(D☐) Aucune des réponses précédentes

Q17 . Le corps de la boucle « `i = 0; while (i = 0) { corps; i ++; }` » s'exécute :

(A☒) 0 fois
(B☐) 1 fois
(C☐) 2 fois
(D☐) Aucune des réponses précédentes

Q18 . Le corps de la boucle « `i = 1; while (++ i < 10)` *corps*`;` » s'exécute :

(A☐) 7 fois
(B☒) 8 fois
(C☐) 9 fois
(D☐) 10 fois

Q19 . Le corps de la boucle « `i = 8; while (i /= 2, (i % 2) == 0)` *corps*`;` » s'exécute :

(A☐) 0 fois
(B☒) 2 fois
(C☐) 3 fois
(D☐) Aucune des réponses précédentes

Q20 . Le corps de la boucle « `i = 9; do` *corps*`; while (++ i < 10);` » s'exécute :

(A☒) 1 fois
(B☐) 2 fois
(C☐) 3 fois
(D☐) 4 fois

Q21 . Le corps de la boucle « `i = 0; do` *corps*`; while (i ++ < 10);` » s'exécute :

(A☐) 1 fois
(B☐) 9 fois
(C☐) 10 fois
(D☒) 11 fois

Q22 . Le corps de la boucle imbriquée « `for (i = 0; i < 3; i ++) for (j = 0; j <= i; j ++)` *corps*`;` » s'exécute :

(A☐) 3 fois
(B☐) 5 fois
(C☒) 6 fois
(D☐) 10 fois

Q23 . Si `a` et `b` sont définis par « `int a = 0x01011100;` » et « `int b = 0x11010001;` », alors « `(a & b)` » vaut :

(A☐) `0x11011101`
(B☐) `0x01011001`
(C☒) `0x01010000`
(D☐) `0x00000001`

Q24 . Si `a` et `b` sont définis par « `int a = 0x11000101;` » et « `int b = 0x01101110;` », alors « `(a | b)` » vaut :

(A☒) `0x11101111`
(B☐) `0x01000100`
(C☐) `0x00000001`
(D☐) Aucune des réponses précédentes

Q25 . Si `a` et `b` sont définis par « `int a = 0x00100111;` » et « `int b = 0x00100110;` », alors « `(a ^ b)` » vaut :

(A☐) `0x11011000`
(B☒) `0x00000001`
(C☐) `0x00100111`
(D☐) `0x00100110`

Q26 . Si `a` et `b` sont définis par « `int a = 0x10001011;` » et « `int b = 0x10110110;` », alors « `(a & ~b)` » vaut :

(A☐) `0x11110110`
(B☒) `0x00001001`
(C☐) `0x00011001`
(D☐) `0x11100110`

Q27 . Si `a` et `b` sont définis par « `int a = 0x01000101;` » et « `int b = 0x00000000;` », alors « `(a && b)` » vaut :

(A☐) `0x01000101`
(B☐) `0x00000001`
(C☒) `0x00000000`
(D☐) Aucune des réponses précédentes

Q28 . Si `i` est défini par « `int i;` », alors « `('A' + i)` » est de type :

(A☐) `(char)`
(B☐) `(short)`
(C☒) `(int)`
(D☐) `(long)`

Q29 . Si `i` est défini par « `int i;` », alors « `(i * 2.0F)` » est de type :

(A☐) `(int)`
(B☐) `(long)`
(C☒) `(float)`
(D☐) `(double)`

Q30 . Si `i` est défini par « `int i;` », alors « `((float) i / 2.0)` » est de type :

(A☐) `(int)`
(B☐) `(long)`
(C☐) `(float)`
(D☒) `(double)`

Q31 . Si `i` est défini par « `int i;` », alors « `i *= 2.0` » est de type :

(A☒) `(int)`
(B☐) `(long)`
(C☐) `(float)`
(D☐) `(double)`

Q32 . Si `i` et `d` sont définis par « `int i; double d;` », alors « `(a >= 0) ? i : d` » est de type :

(A☐) `(int)`
(B☒) `(double)`
(C☐) `(long)`
(D☐) Cela dépend

Q33 . Si `i` et `d` sont définis par « `int i; double d;` », alors « `(int) i % d` » est de type :

(A☐) `(int)`
(B☐) `(double)`
(C☐) `(long)`
(D☒) C'est une expression erronée

Q34 . Si `t` est défini par « `int * t;` », alors « `&(t[2] + 1)` » est de type :

(A☐) `(int *)`
(B☐) `(int **)`
(C☐) `(int)`
(D☒) C'est une expression erronée

Q35 . Si `p` est défini par « `int * p;` », alors « `&p[1] + 1` » est de type :

(A☐) `(int)` (C☐) `(int **)`
(B☒) `(int *)` (D☐) C'est une expression erronée

Q36 . Si `t` est défini par « `int * t;` », alors « `t[*t]` » est de type :

(A☒) `(int)` (C☐) `(int **)`
(B☐) `(int *)` (D☐) C'est une expression erronée

Q37 . Si `t` est défini par « `int * t;` », alors « `&t` » est de type :

(A☐) `(int)` (C☒) `(int **)`
(B☐) `(int *)` (D☐) C'est une expression erronée

Q38 . Si `t` est défini par « `int t[2];` », alors « `&t` »est de type :

(A☐) `(int)` (C☐) `(int **)`
(B☐) `(int *)` (D☒) C'est une expression erronée

Q39 . Si `t` est défini par « `int t[2];` », alors « `&t[2] + 1` » est de type :

(A☐) `(int)` (C☐) `(int **)`
(B☒) `(int *)` (D☐) C'est une expression erronée

Q40 . Si `t` est défini par « `int * t;` », alors « `&(t[2] + 1)` » est de type :

(A☐) `(int)` (C☐) `(int **)`
(B☐) `(int *)` (D☒) C'est une expression erronée

Q41 . Si `t` est défini par « `int t[] = { 1, 0 };` », alors « `t[t[*(t + 1)]]` » vaut :

(A☒) `0` (C☐) `t`
(B☐) `1` (D☐) C'est une expression erronée

Q42 . Si `t` est défini par « `int t[] = { 1, 0 };` », alors « `t[*(t + *t)]` » vaut :

(A☐) `0` (C☐) `t`
(B☒) `1` (D☐) C'est une expression erronée

Q43 . Si `t` est défini par « `int t[2][3] = { { 3, 4, 5 }, { 0, 1, 2 } };` », alors « `t[0][4]` vaut :

(A☐) `0` (C☐) `5`
(B☒) `1` (D☐) C'est une expression erronée

Q44 . Si `t` est défini par « `int t[] = { 1, 0 };` », alors « `*(*t + 1)` » est de type :

(A☐) `(int)` (C☐) `(int **)`
(B☐) `(int *)` (D☒) C'est une expression erronée

Q45 . Si `t` est défini par « `int * t;` », alors « `&t` » est de type :

(A☐) `(int)`
(B☐) `(int *)`
(C☒) `(int **)`
(D☐) C'est une expression erronée

Q46 . Si `t` est défini par « `int t[] = { 1, 0 };` », alors « `&t` » est de type :

(A☐) `(int)`
(B☐) `(int *)`
(C☐) `(int **)`
(D☒) C'est une expression erronée

Q47 . Si `t` est défini par « `int t[] = { 1, 0 };` », alors « `*t` » est de type :

(A☒) `(int)`
(B☐) `(int *)`
(C☐) `(int **)`
(D☐) C'est une expression erronée

Q48 . Si `s` et `t` sont définis par « `int t[] = { 1, 2, 3, 4, 5 }; int * s = t + 4` », alors « `s[-2]` vaut :

(A☐) `1`
(B☒) `3`
(C☐) `4`
(D☐) C'est une expression erronée

Q49 . Si `Brol_` est définie par « `struct Brol_ { int a; double d; };` », alors `sizeof (Brol_)` vaudra :

(A☒) Toujours au moins `(sizeof(int) + sizeof(double))`
(B☐) Toujours exactement `(sizeof(int) + sizeof(double))`
(C☐) Toujours au plus `(sizeof(int) + sizeof(double))`
(D☐) Aucune des réponses précédentes

Q50 . Si `Brol_` est définie par « `union Brol_ { int a; double d; };` », alors `sizeof (Brol_)` vaudra :

(A☐) Toujours au moins `(sizeof(int) + sizeof(double))`
(B☐) Toujours `sizeof(int)`
(C☒) Toujours `sizeof(double)`
(D☐) Aucune des réponses précédentes

Q51 . Si un pointeur `p` sur une structure `Brol_` est défini par « `struct Brol_ { int a; struct { double x, y; } t; } *p;` », alors le champ `x` de la structure `Brol_` pointée par `p` est accédé par l'expression :

(A☐) `p.t.x`
(B☐) `p->a.x`
(C☐) `p->t->x`
(D☒) `p->t.x`

Q52 . Si `o1` et `o2` sont deux variables de type structuré `Brol_`, définis par « `struct Brol_ { int a; struct { double x, y; } t; } o1, o2;` », alors la copie de tous les champs de `o2` dans les champs de `o1` s'écrit :

(A☐) `o1.* = o2.*;`
(B☒) `o1 = o2;`
(C☐) `o1 <<= o2;`
(D☐) `o1 .= o2;`

Q53 . Si `p` est un pointeur sur un tableau de caractères, alors `p` peut se définir comme :

(A☒) `char * p`
(B☐) `char * p []`
(C☐) `(char []) p *`
(D☐) `char ** p`

Q54 . Si `p` est un pointeur sur un tableau de `double`, alors `p` peut s'écrire :

(A☐) `double ** p`
(B☒) `double * p`
(C☐) `double (* p)[]`
(D☐) `double * p[]`

Q55 . Si `p` est un pointeur sur un tableau de pointeurs sur des tableaux d'entiers, alors `p` peut s'écrire :

(A☐) `int (* (* p)[])[]`
(B☒) `int ** p`
(C☐) `int [][] *p`
(D☐) `int *** p`

Q56 . Si `f` est une fonction prenant en paramètre un entier et ne renvoyant rien, alors le prototype de `f` peut s'écrire :

(A☒) `void f (int)`
(B☐) `f (int -> void)`
(C☐) `int f (void)`
(D☐) `void (* f) (int)`

Q57 . Si `f` est une fonction prenant en paramètre un pointeur sur un tableau de `double` et renvoyant un pointeur d'entier, alors le prototype de `f` peut s'écrire :

(A☐) `int f (double [])`
(B☐) `int * f ([] double)`
(C☒) `int * f (double *)`
(D☐) `int (* f) (double [])`

Q58 . Pour inclure le fichier d'en-tête système « `brol.h` », on doit écrire :

(A☒) `#include <brol.h>`
(B☐) `#include 'brol.h'`
(C☐) `#include "brol.h"`
(D☐) `#include brol.h`

Q59 . Si l'on inclut le fichier d'en-tête « `brol.h` » au moyen de la directive « `#include "brol.h"` », le fichier sera cherché :

(A☐) Dans les répertoires système seulement
(B☐) Dans les répertoires système puis dans les répertoires spécifiés par l'utilisateur
(C☐) Dans les répertoires spécifiés par l'utilisateur puis dans les répertoires système
(D☒) Dans les répertoires spécifiés par l'utilisateur seulement

Q60 . Pour définir une macro `PI` égale à π à la cinquième décimale et utilisable dans des expressions mathématiques, on écrit :

(A☐) `#define PI 3,14159;`
(B☒) `#define PI 3.14159`
(C☐) `#define PI = 3.14159;`
(D☐) `#define PI 3.14159;`

Q61 . Pour définir une macro `PRODUIT` qui calcule le produit des deux arguments qui lui sont passés, quelles que soient les expressions qui lui sont passées, on écrit :

(A☐) `#define PRODUIT (a,b) ((a)*(b))`
(B☐) `#define PRODUIT(a,b) (a*b)`
(C☒) `#define PRODUIT(a,b) ((a)*(b))`
(D☐) `#define PRODUIT(a,b) ((a)*(b));`

Q62 . Pour n'inclure le fragment de code *frag* que si la macro `BROL` est définie, il ne faut pas écrire :

(A☐) `#if defined BROL`
frag
`#endif`

(B☒) `#if defined BROL`
`#else`
frag
`#endif`

(C☐) `#ifdef BROL`
frag
`#endif`

(D☐) `#ifndef BROL`
`#else`
frag
`#endif`

Q63 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L1 est :

(A☐) 0
(B☒) 1
(C☐) 2
(D☐) 3

Q64 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L2 est :

(A☒) 0
(B☐) 1
(C☐) 2
(D☐) 3

Q65 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L3 est :

(A☒) 0
(B☐) 1
(C☐) 2
(D☐) 3

Q66 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L4 est :

(A☒) 0
(B☐) 1
(C☐) 2
(D☐) 3

Q67 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L5 est :

(A☐) 0
(B☒) 1
(C☐) 2
(D☐) 3

Q68 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L6 est :

(A☐) 0
(B☐) 1
(C☐) 2
(D☒) 3

Q69 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L7 est :

(A☐) 1
(B☐) 3
(C☒) 4
(D☐) 5

Q70 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L8 est :

(A☐) 6
(B☐) 7
(C☐) 9
(D☒) 10

Q71 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L9 est :

(A☐) 1
(B☒) 3
(C☐) 5
(D☐) 7

Q72 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L10 est :

(A☐) 2
(B☐) 4
(C☒) 5
(D☐) 8

Q73 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L11 est :

(A☐) 7
(B☒) 10
(C☐) 12
(D☐) 13

Q74 . Dans le **Programme 1** situé à la fin du test, la valeur affichée après L12 est :

(A☒) 3
(B☐) 6
(C☐) 7
(D☐) 9

Programme 1

```
static int a = 2;
int        b = 3;
int f1 (int i) { return (i + 1); }
int f2 (int i) { return (i ++); }
int f3 (int * i) { return ((*i) ++); }
int f4 (int a)
{
  return (a ++ + b ++);
}
int f5 (int * a, int c)
{
  static int d = 1;
  int        e = 0;

  return ((*a) ++ + c + d ++)
}
int f6 (int * b, int c)
{
  int d = 1;

  return ((*b) ++ + c + d ++)
}
main ()
{
  int e = 0;
  printf ("L1:  %d\n", f1 (e));
  printf ("L2:  %d\n", f2 (e));
  printf ("L3:  %d\n", f2 (e));
  e = 0;
  printf ("L4:  %d\n", f3 (&e));
  printf ("L5:  %d\n", f3 (&e));
  e = 0;
  printf ("L6:  %d\n", f4 (e));
  printf ("L7:  %d\n", f4 (e));
  printf ("L8:  %d\n", f4 (b));
  e = 0;
  printf ("L9:  %d\n", f5 (&a, e));
  printf ("L10: %d\n", f5 (&e, a));
  printf ("L11: %d\n", f6 (&b, a));
  printf ("L12: %d\n", f6 (&e, e));
}
```