

Nom :	Année scolaire :	2006–2007
Prénom :	Date :	26 juin 2007

Module INF423
Session de juin

Programmation avancée en C

Contrôle de connaissance¹ de 45 minutes

 ERCI de répondre (au moins) dans les blancs.
Lire tout le sujet en entier du début à la fin, en commençant à la première page et jusqu'à la dernière page, avant de commencer à répondre : cela peut vous donner de l'inspiration et vous permettre de mieux allouer votre temps en fonction de vos compétences.

Chaque question sera notée entre 0 et 10 et la note globale sera calculée par une fonction des notes élémentaires. La fonction définitive sera choisie après correction des copies.

Attention : tout ce que vous écrirez sur cette copie pourra être retenu contre vous, voire avoir une influence sur la note d'INF423.

1 Généralités

Question 1 : Quelle valeur le programme suivant affiche lors de l'exécution ?

```
1 #include <stdio.h>
   #include <stdlib.h>
3
   int *_fonction(int valeur){
5     valeur = valeur + 3;
     return &valeur;
```

¹Avec document, sans triche, sans copie sur les voisins, sans micro-ordinateur portable ou non, sans macro-ordinateur, sans téléphone portable ou non, sans oreillette de téléphone ni de dictaphone, sans talkie-walkie, sans télépathie, sans métépsychose, sans pompe. Sont tolérés : anti-sèche, tatouage ou vêtement imprimé en rapport avec le sujet, mouchoir de poche pré-imprimé, piercing ou scarification en rapport avec l'INF423, bronzage à code barre ou 2D...

```
7 }

9 int _main (int _argc , _char ** argv) _{
    _int _valeur = 1;
11    _fonction ( valeur );

13    _printf ( "%d\n" , _valeur );
    _exit (0);
15 }
```

(1 minute) ☐

→

Réponse possible :

Le programme affiche la valeur 1. En effet, à l'intérieur de la fonction main, valeur est définie à 1, puis la fonction fonction est appelé. Lors de cet appel, une copie de la valeur de la variable valeur est effectuée et est passée à la fonction. Ainsi, cette fonction ne peut pas modifier la valeur de la variable valeur. Le printf affiche ensuite la valeur de la variable valeur qui n'a pas pu être modifiée par la fonction fonction et donc qui est resté à sa valeur initiale, c'est-à-dire 1.

Question 2 : Quelle valeur affiche le programme suivant lors de l'exécution ?

```
1 #include <stdio.h>
  #include <stdlib.h>

3
  int _fonction (int _param) _{
5     _int _valeur = 0;
    _valeur = _valeur + 3;
7     _return (param + _valeur );
    }

9
  int _main (int _argc , _char ** argv) _{
11     _int _valeur = 1;
    _fonction ( valeur );

13
    _printf ( "%d\n" , _valeur );
15     _exit (0);
    }
```

(1 minute) ☐

→

Réponse possible :

Le programme affiche la valeur 1 pour les mêmes raisons que la question précédente.

Une copie de valeur est réalisée lors de l'appel de la fonction `Fonction` et donc cette dernière ne peut pas influencer sur la valeur de la variable dans la fonction `main`.

Question 3 : Que se passe-t-il lors de l'exécution du programme suivant² ?

```

1 #include <stdio.h>
2 #include <stdlib.h>
  #include <string.h>
4
  int main(int argc, char** argv){
6     char* chaine;
      chaine = strdup("Test");
8
      printf("%d\n", (int)chaine[4]);
10 }

```

(2 minutes) ☐

→
→
→
→
→

Réponse possible :

Le programme affiche le chiffre 0. L'appel à la fonction `strdup` renvoie un pointeur pointant vers une zone mémoire contenant les caractères 'T', 'e', 's', 't' suivis du caractère `\0` dont la valeur est 0, comme pour toute chaîne de caractères en C. `chaine[4]` représente donc le cinquième caractère (la numérotation des indices d'un tableau commence toujours à 0) et donc `\0`. La fonction `printf` affiche la valeur de ce caractère et donc affiche 0.

Question 4 : Qu'affiche le programme suivant ?

```

1 #include <stdio.h>
2 #include <stdlib.h>
4
  int main(int argc, char* argv[]){
      printf("%d, %d, %d\n", 0x15, 016, 17);
6      exit(0);
  }

```

(3 minutes) ☐

→

²Si vous ne vous souvenez plus de la fonction la plus importante du langage C, faite un détour par la question 6. Mais à la page 1 on vous avait bien dit de lire tout le sujet...

Réponse possible :

Le programme affiche 21, 14, 17. 0x devant un nombre indique que celui-ci est exprimé en hexadécimal (base 16), et un 0 devant un nombre indique que celui-ci est exprimé en octal (base 8). Par contre, le %d de la fonction printf affiche ces nombres en décimal.

2 Entiers et chaînes de caractères

Question 5 : Écrivez une fonction qui prend en paramètre un entier (`int`) et qui retourne la chaîne de caractères contenant la valeur exprimée en base 10 de cet entier. Vous ne devez pas utiliser les fonctions de la famille `printf` (`sprintf`, `aprinf`, etc.), `itoa` ou autre qui ferait déjà trivialement le travail. (10 minutes) ☐

→

Réponse possible :

```

1  char *_int_to_char ( int value ) {
    int size = 0;
3  int temp = value;
    int index;
5  char *result;

7  while ( temp != 0 ) {
    temp /= 10;
9  size++;
    }

11
    if ( value < 0 ) size++;
13 if ( size == 0 ) size++;

15 result = malloc ( size + 1 );
    result [ size ] = 0;

17
    index = size - 1;
19 temp = value;
    if ( value < 0 ) temp = -value;

21
    while ( temp != 0 ) {
23 result [ index ] = '0' + ( temp % 10 );
        temp /= 10;
25 index--;
    }

27
    if ( value < 0 ) result [ 0 ] = '-';

29
    return result;
31 }

```

3 strdup

Question 6 : Écrivez un équivalent de la fonction `strdup` (sans l'utiliser bien entendu). Cette fonction prend en paramètre une chaîne de caractères, la duplique et renvoie la chaîne dupliquée (il est ainsi possible de modifier la chaîne originale et la chaîne dupliquée indépendamment l'une de l'autre). (10 minutes) □

→

4 Endianness (ou le sexe des octets)

Sur la très grande majorité des processeurs actuels d'ordinateurs grand public, un entier (type `int`) est stocké sur 4 octets (32 bits). La mémoire peut, quant à elle, stocker, pour chaque adresse, un octet. Se pose donc la question de savoir dans quel ordre stocker ces quatre octets en mémoire. Cet ordre est appelé *endianness* et plusieurs religions³ se sont formées. Il est très important de connaître ce point, par exemple lorsque l'on veut faire de la communication entre plusieurs machines potentiellement basées sur des architectures différentes.

Étudions le stockage de la valeur `0x12345678` à l'adresse i en mémoire.

Cas *big-endian* (appelé parfois gros-boutiste en Français) : l'octet de poids fort est stocké à l'adresse la plus basse. C'est le cas par exemple des processeurs IBM Cell de la PlayStation/3, le 68000 de MOTOROLA ou des SPARC de SUN.

0x78	$i + 3$
0x56	$i + 2$
0x34	$i + 1$
0x12	i

Cas *little-endian* (appelé parfois petit-boutiste en Français) : l'octet de poids fort est stocké à l'adresse la plus élevée. C'est le cas notamment de l'architecture très répandue INTEL *x86*.

0x12	$i + 3$
0x34	$i + 2$
0x56	$i + 1$
0x78	i

Certaines architectures supportent les deux conventions et sont appelées *big-endian*. C'est le cas notamment des architectures IBM POWERPC et INTEL IA-64.

Enfin, certaines architectures plus exotiques sont MIDDLE-ENDIAN. Exemple du PDP-11 :

³Et donc les guerres qui vont avec...

0x34	$i + 3$
0x12	$i + 2$
0x78	$i + 1$
0x56	i

Question 7 : Écrivez tout d'abord une fonction permettant de détecter sur quelle type d'architecture elle s'exécute (*big-endian* ou *little-endian*). (6 minutes) ☐

→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→

Réponse possible :

```

1 void detect_endianness(void) {
2     unsigned int i = 0x12345678;

4     unsigned char *ichar = (unsigned char *)&i;

6     if (*ichar == 0x12) printf("Big_endian");
       if (*ichar == 0x78) printf("Little_endian");
8 }

```

Question 8 : Supposons que l'architecture sur laquelle vous travaillez soit *big-endian*. Vous avez lu, octet par octet, un entier (codé sur quatre octets) depuis un fichier où il était exprimé suivant la représentation *little-endian*. Donc si vous lisez l'entier simplement en mémoire, sa valeur sera fausse... Écrivez une fonction permettant de récupérer sa valeur correcte depuis la

mémoire.

(10 minutes) ☐

Réponse possible :

```

1  int little_to_big_endian(int little) {
2      int result;

4      unsigned char *littlechar = (unsigned char *) &little;
      unsigned char *bigchar = (unsigned char *) &result;

6      bigchar[0] = littlechar[3];
      bigchar[1] = littlechar[2];
      bigchar[2] = littlechar[1];
      bigchar[3] = littlechar[0];

12     return result;
    }

```

Durée totale estimée : 43 minutes.