Travaux dirigés 1 : Assembleur, affectation en C, if else.

1 Langage machine

L'objectif de cette première partie est de vous familiariser avec le cycle d'exécution d'un processeur et avec la notion de flux d'instructions. Pour cela, il vous est demandé d'écrire de petits programmes dans le langage assembleur `amil` (*assembleur miniature pour l'informatique de licence*) et de simuler leur exécution soit à la main en écrivant une trace soit en utilisant le simulateur en ligne : `http://www-lipn.univ-paris13.fr/~boudes/amilweb/`.

Correction.

Note aux chargés de TD.

- *Dans le texte des corrections se trouvent également quelques explications de correction. Bien entendu, vous n'avez pas à en parler aux étudiants (boucle for,* `gcc -S`*, etc.).*
- *Amil est très rudimentaire dans ses possibilités d'écrire des commentaires. Dans cette feuille, il y a des commentaires ligne à ligne, mais aussi une tentative d'indiquer où sont les différents blocs de code dans un style balise xml/html : une ligne avec en commentaire* `<bloc1>` *signale la première ligne du bloc* `bloc1`*, une ligne avec en commentaire* `</bloc1>` *signale la dernière ligne du bloc, et une ligne avec en commentaire* `<bloc1/>` *signale l'unique ligne du bloc* `bloc1`*.*
- *les traces contiennent une colonne* instructions *: cette colonne n'a pas lieu d'être dans les corrections (elle aide juste à relire la trace, qui est générée automatiquement).*

Jeu d'instructions (simplifié)

`stop`	Arrête l'exécution du programme.
`noop`	N'effectue aucune opération.
`saut i`	Met le compteur de programme à la valeur i.
`sautpos ri j`	Si la valeur contenue dans le registre i est positive ou nulle, met le compteur de programme à la valeur j.
`valeur x ri`	Initialise le registre i avec la valeur x.
`lecture i rj`	Charge, dans le registre j, le contenu de la mémoire d'adresse i.
`ecriture ri j`	Écrit le contenu du registre i dans la mémoire d'adresse j.
`inverse ri`	Inverse le signe du contenu du registre i.
`add ri rj`	Ajoute la valeur du registre i à celle du registre j (la somme obtenue est placée dans le registre j).
`soustr ri rj`	Soustrait la valeur du registre i à celle du registre j (la différence obtenue est placée dans le registre j).
`mult ri rj`	Multiplie par la valeur du registre i celle du registre j (le produit obtenu est placé dans le registre j).

1.1 Écriture de programmes simples

Écrire les programmes répondant aux problèmes suivants :

1. Soit la valeur 8 contenue dans la case mémoire d'adresse 10. Recopier cette valeur à l'adresse 11.

Cycles	CP	instruction	r0	10	11
INIT	1		?	5	?
1	2	lecture 10 r0	5		
2	3	ecriture r0 11			5
3	4	stop			

FIGURE 1 – Simulation de la copie de valeur

Correction.

```
1        lecture 10 r0
2        ecriture r0 11
3        stop
4
```

2. Soit une valeur quelconque, a, contenue à l'adresse 10. Écrire la valeur de $a \times 2 + 1$ dans la case d'adresse 11.

Correction.

```
1        lecture 10 r0
2        valeur 2 r1
3        mult r1 r0
4        valeur 1 r1
5        add r1 r0
6        ecriture r0 11
7        stop
8
```

1.2 Trace de programme assembleur

Nous allons désormais produire une représentation de l'exécution pas à pas de nos programmes. Une *trace* d'un programme assembleur sera un tableau dont chaque ligne correspond à l'exécution d'une instruction par le processeur. Une colonne contiendra le cycle d'horloge du processeur et une autre colonne le compteur de programme. Il y aura également une colonne par registre utilisé dans le programme et par case mémoire où des données sont lues ou écrites par les instructions du programme. Une première ligne, d'initialisation, montrera l'état de ces registres et cases mémoires avant le début du programme. Ensuite, chaque exécution d'instruction sera représentée par une nouvelle ligne du tableau, jusqu'à exécution de l'instruction `stop`. Dans cette ligne nous représenterons le cycle d'horloge du processeur (en commençant à 0 pour la ligne d'initialisation), la valeur du compteur de programme après exécution de l'instruction, et, s'il y a lieu, la valeur du registre ou de la case mémoire modifiée par le programme.

1.2.1 Exemple de trace et première trace

Soit le programme ci-dessous :

```
1   lecture 10 r0
2   valeur 5 r2
3   soustr r2 r0
4   sautpos r0 8
5   valeur 0 r0
6   ecriture r0 11
7   saut 9
8   ecriture r0 11
9   stop
10  14
11  ?
```

On représentera alors sa trace comme ceci :

Instructions	Cycles	CP	r0	r2	10	11
Initialisation	0	1	?	?	14	?
`lecture 10 r0`	1	2	14			
`valeur 5 r2`	2	3		5		
`soustr r2 r0`	3	4	9			
`sautpos r0 8`	4	**8**				
`ecriture r0 11`	5	9				9
`stop`	6	10				

La colonne *Instructions* n'est pas nécessaire, elle sert juste ici à la compréhension, sans cette colonne le mot-clé *initialisation* pourra être placé dans la colonne *Cycles*. Remarquez par exemple qu'il n'y a pas de colonne pour le registre 1, puisque celui-ci n'est pas utilisé.

1. Refaire la trace du programme où la valeur 14 à l'adresse 10 est remplacée par 2.
2. Quelles sont les valeurs contenues dans les registres 0, 1 et 2 après arrêt sur `stop` ?

Correction.

Instructions	*Cycles*	*CP*	*r0*	*r2*	*10*	*11*
Initialisation	*0*	*1*	*?*	*?*	*14*	*?*
`lecture 10 r0`	*1*	*2*	*14*			
`valeur 5 r2`	*2*	*3*		*5*		
`soustr r2 r0`	*3*	*4*	*9*			
`sautpos r0 8`	*4*	***8***				
`ecriture r0 11`	*5*	*9*				*9*
`stop`	*6*	*10*				

La valeur contenue dans le registre 0 est 0, celle contenue dans le registre 1 est indéterminée, celle contenue dans le registre 2 est 5.

1.3 Exécution conditionnelle d'instructions

À l'aide de l'instruction `sautpos`, écrire les programmes correspondant aux algorithmes suivants et les exécuter sur un exemple (trace ou simulateur) afin de tester leur correction :

1. Soient la valeur a à l'adresse 15, b à l'adresse 16. Si $a >= b$ alors écrire a à l'adresse 17 sinon écrire b à l'adresse 17.

Correction.

```
1   lecture 15 r0 # a
2   lecture 16 r1 # b
3   soustr r1 r0  # r0 vaut a - b
4   sautpos r0 8  # si a - b >= 0 (cad a >= b) on saute sur le alors
5   lecture 16 r0 #  sinon ecrire b a l'adresse 17
6   ecriture r0 17
7   saut 10
8   lecture 15 r0 # alors ecrire a a l'adresse 17
9   ecriture r0 17
10  stop
11  ?
12  ?
13  ?
14  ?
15  23 # a
```

Instructions	Cycles	CP	r0	r1	15	16	17
Initialisation	0	1	?	?	23	45	?
`lecture 15 r0`	1	2	23				
`lecture 16 r1`	2	3		45			
`soustr r1 r0`	3	4	-22				
`sautpos r0 8`	4	5					
`lecture 16 r0`	5	6	45				
`ecriture r0 17`	6	7					45
`saut 10`	7	**10**					
`stop`	8	11					

FIGURE 2 – Simulation du calcul du max de a et b (1)

Instructions	Cycles	CP	r0	r1	15	16	17
Initialisation	0	1	?	?	45	23	?
`lecture 15 r0`	1	2	45				
`lecture 16 r1`	2	3		23			
`soustr r1 r0`	3	4	22				
`sautpos r0 8`	4	**8**					
`lecture 15 r0`	5	9	45				
`ecriture r0 17`	6	10					45
`stop`	7	11					

FIGURE 3 – Simulation du calcul du max de a et b (2)

```
16   45 # b
17   ?  # <-- valeur de sortie
```

---

2. Soit l'âge d'une personne à l'adresse 15. Si cette personne est majeure alors écrire 1 à l'adresse 16 sinon écrire 0 à l'adresse 16.

Correction.

```
1   lecture 15 r0 # si age >= 18
2   valeur -18 r1
3   add r0 r1  # r1 vaut age - 18
4   sautpos r1 8
5   valeur 0 r0 # sinon ecrire 0 a l'adresse 16
6   ecriture r0 16
7   saut 10
8   valeur 1 r0 # alors ecrire 1 a l'adresse 16
9   ecriture r0 16
10  stop
11  ?
12  ?
13  ?
14  ?
15  17 # a
16  ?  # <-- valeur de sortie
```

---

3. Soient trois cases mémoires contenant trois entiers, a, b et c. Calculer le minimum de ces trois entiers et l'écrire dans une autre case mémoire. *Commencer par réflechir à l'algorithme.*

Correction.

Cycles	CP	instruction	r0	r1	15	16
INIT	1		?	?	17	?
1	2	lecture 15 r0	17			
2	3	valeur -18 r1		-18		
3	4	add r0 r1		-1		
4	5	sautpos r1 8				
5	6	valeur 0 r0	0			
6	7	ecriture r0 16				0
7	**10**	saut 10				
8	11	stop				

FIGURE 4 – Simulation du test de la majorité (1)

Cycles	CP	instruction	r0	r1	15	16
INIT	1		?	?	20	?
1	2	lecture 15 r0	20			
2	3	valeur -18 r1		-18		
3	4	add r0 r1		2		
4	**8**	sautpos r1 8				
5	9	valeur 1 r0	1			
6	10	ecriture r0 16				1
7	11	stop				

FIGURE 5 – Simulation du test de la majorité (2)

– *Soient a, b et c trois entiers*

```
20   12 # a
21   23 # b
22   6  # c
23   ?  # min
```

– *min est initialise à a (par défaut)*

```
 1   lecture 20 r0
 2   ecriture r0 23
```

– *Si b < min alors min vaut b*

```
 3   lecture 21 r0
 4   lecture 23 r1
 5   inverse r1
 6   add r0 r1    # r1 vaut b - min
 7   sautpos r1 9
 8   ecriture r0 23
```

– *Si c < min alors min vaut c*

```
 9   lecture 22 r0
10   lecture 23 r1
11   inverse r1
12   add r0 r1 # r1 vaut c - min
13   sautpos r1 15
14   ecriture r0 23
```

– *min contient le minimum de a, b et c*

```
15   stop
```

1.4 Boucles infinies

Avec l'instruction `saut`, écrire un programme qui ne termine jamais.

Correction.

```
1   saut 1
2   stop # <- jamais atteint
```

---

2 Premier programme C et affectation

Programme C

```
 1  /* Declaration de fonctionnalites supplementaires */
 2  #include <stdlib.h> /* EXIT_SUCCESS */
 3
 4  /* Declaration des constantes et types utilisateurs */
 5
 6  /* Declaration des fonctions utilisateurs */
 7
 8  /* Fonction principale */
 9  int main()
10  {
11      /* Declaration et initialisation des variables */
12      int x = 5;
13      int y;
14
15      y = 2;
16      x = y;
17
18      /* valeur fonction */
19      return EXIT_SUCCESS;
20  }
21
22  /* Definitions des fonctions utilisateurs */
```

Traduction

```
 1  valeur 2 r0
 2  ecriture r0 11
 3  lecture 11 r0
 4  ecriture r0 10
 5  stop
 6
 7
 8
 9
10  5
11  ?
```

FIGURE 6 – Un programme C et sa traduction machine

Voici, figure 6, un premier programme C. Nous donnons ici la traduction de ce code source en amil. Il s'agit d'un artifice pédagogique, la traduction réelle en code binaire exécutable est plus compliquée. Par analogie avec la musique, le source est la partition, et le fichier exécutable est le morceau musical (codé sur le support adapté au système de lecture : un fichier mp3, un CD, etc.). La traduction est effectuée par un ensemble de programmes, le source doit donc obéir à des règles syntaxiques précises.

Correction. *Pour la correction, voici également (figure 7) les traces de ces deux programmes, auxquelles se référer si les étudiants ne comprennent pas les programmes en eux même.*

Les textes entre `/*` et `*/` sont des *commentaires*, ils ne feront pas partie du programme exécutable, ils servent aux humains qui manipulent les programmes. Les commentaires du programme figure 6 vous serviront au cours du semestre à structurer tous vos programmes C.

Tout programme C comporte une fonction principale, le `main()`, qui sert de point d'entrée au programme. Cette fonction doit se terminer par l'instruction `return EXIT_SUCCESS`. En renvoyant cette valeur, le `main` signale au système d'exploitation la terminaison correcte du programme.

L'instruction `int x = 5` déclare une variable `x` et fixe sa valeur initiale à 5. Le mot clé `int` signifie que cette variable contiendra un entier. Dans le code amil `x` correspond à l'adresse `10` où se trouve intialement la valeur `5`.

L'instruction `int y` déclare une variable entière `y` sans l'initialiser. L'effet de cette déclaration est de réserver un espace mémoire pour y stocker un entier.

Le signe égal (`=`) a un sens bien particulier, il dénote une *affectation*. L'objectif de ce TD est de bien comprendre l'affectation. La partie à gauche du signe égal doit désigner une case mémoire, c'est typiquement une variable. La partie à droite du signe égal est une expression

Ligne	x	y	sortie
initialisation	5	?	
15		2	
16	2		
19	renvoie `EXIT_SUCCESS`		

(a) Trace en C

Instructions	Cycles	CP	r0	10	11
Initialisation	0	1	?	5	?
`valeur 2 r0`	1	2	2		
`ecriture r0 11`	2	3			2
`lecture 11 r0`	3	4	2		
`ecriture r0 10`	4	5		2	
`stop`	5	6			

(b) Trace amil

FIGURE 7 – Traces

dont la valeur sera évaluée et écrite à l'adresse à laquelle renvoie la partie gauche. Par exemple, `y = 2` a été traduit en code machine par une instruction évaluant l'expression `2` dans un registre (ici `valeur 2 r0`), et par une instruction d'écriture de la valeur trouvée dans la mémoire réservée à `y`. Une variable s'évalue comme sa valeur (celle contenue dans la mémoire correspondante, au moment de l'évaluation).

2.1 Questions

1. Quel espace mémoire a été réservé pour `y` dans le code amil ?

   Correction. *L'adresse 11.*

2. Comment a été traduite l'instruction d'affectation `x = y` en amil ?

   Correction. *Lignes 3 et 4 :*

   ```
   lecture 11 r0
   ecriture r0 10
   ```

---

3. Si il y avait `y = x + 2`, ligne 15 dans le programme C, à la place de `y = 2`, quel serait le code amil correspondant ? Et `x = x + 1` ligne 16 ?

   Correction. *Ne pas se préoccuper du décalage des lignes de code amil. Pour* `y = x + 2` *:*

   ```
   lecture 10 r0
   valeur 2 r1
   add r1 r0
   ecriture r0 11
   ```

   Pour `x = x + 1` *:*

   ```
   lecture 10 r0
   valeur 1 r1
   add r1 r0
   ecriture r0 10
   ```

2.2 Échange de valeurs : introduction du problème

Nous avons deux tableaux anciens, chacun accroché à un clou, et un troisième clou, libre, sur le mur d'une exposition. Pour des critères esthétiques, nous voulons changer de place nos deux tableaux sans les mettre par terre, car cela risquerait d'abîmer nos précieuses toiles, et en ne déplaçant qu'une toile à la fois. Comment faire ?

FIGURE 8 – Échanger les deux tableaux en utilisant le clou libre

Correction.

```
/* tableau 1 (clou 1) -> clou 3 */
/* tableau 2 (clou 2) -> clou 1 */
/* tableau 1 (clou 3) -> clou 2 */
```

2.3 Échange des valeurs de deux variables en C

1. Écrire un programme C qui déclare et initialise deux variables entières `x` et `y` et effectue la permutation de ces deux valeurs. Commencer par écrire un algorithme, à l'aide de phrases telles que « Copier la valeur de la variable ... dans la variable ...», en vous inspirant de la question précédente.

   Correction. _ *L'agorithme doit être utilisé pour structurer le code. On ne donne pas ici de valeur à x et y, mais n'hésitez pas à en donner (par initialisation par exemple).*

```
1   /* Declaration de fonctionnalites supplementaires */
2   #include <stdlib.h> /* EXIT_SUCCESS */
3
4   /* Declaration des constantes et types utilisateurs */
5
6   /* Declaration des fonctions utilisateurs */
7
8   /* Fonction principale */
9   int main()
10  {
11      /* Declaration et initialisation des variables */
12      int x = 4;
13      int y = 5;
14      int aux; /* variable auxilliaire pour la permutation */
15
16      /* copie de la valeur de x dans la variable auxilliaire */
17      aux = x;
18      /* copie de la valeur de y dans x */
19      x = y;
20      /* copie dans y de l'ancienne valeur de x, depuis la variable auxilliaire */
21      y = aux;
22
23      /* valeur fonction */
```

Cycles	CP	instruction	r0	10	11	12
INIT	1		?	4	5	?
1	2	lecture 10 r0	4			
2	3	ecriture r0 12				4
3	4	lecture 11 r0	5			
4	5	ecriture r0 10		5		
5	6	lecture 12 r0	4			
6	7	ecriture r0 11			4	
7	8	stop				

FIGURE 9 – Simulation de l'échange de deux valeurs en mémoire (4 et 5)

```
24        return EXIT_SUCCESS;
25    }
26
27    /* Definitions des fonctions utilisateurs */
```

---

2. Traduire ce programme C en un programme amil. On supposera que les deux variables sont stockées aux adresses `10` et `11`.

   Correction. *À nouveau, l'algorithme est à utiliser en commentaires pour structurer le code.*

```
# copie de la case d'adresse 10 dans une case auxillaire (12)
 1 lecture 10 r0
 2 ecriture r0 12
# copie de la case d'adresse 11 à l'adresse 10
 3 lecture 11 r0
 4 ecriture r0 10
# copie de la case auxillaire à l'adresse 11
 5 lecture 12 r0
 6 ecriture r0 11
 7 stop
 9
10 4
11 5
12 ?
```

   Correction.

3. (Facultatif). Donner d'autres solutions en assembleur à ce problème (la permutation des contenus des adresses `10` et `11`).

   Correction. *Par exemple :*

```
# copie de la case d'adresse 10 dans un registre pour sauvegarde
1    lecture 10 r1
# copie de la case d'adresse 11 à l'adresse 10
2    lecture 11 r0
3    ecriture r0 10
# copie de a dans registre de sauvegarde à l'adresse 11
4    ecriture r1 11
5    stop
```

---

4. (Facultatif). Mêmes questions que précédemment mais pour faire une permutation circulaire de 3 valeurs.

Cycles	CP	instruction	r0	r1	10	11
INIT	1		?	?	4	5
1	2	lecture 10 r1		4		
2	3	lecture 11 r0	5			
3	4	ecriture r0 10			5	
4	5	ecriture r1 11				4
5	6	stop				

FIGURE 10 – Simulation de l'échange de deux valeurs en mémoire (4 et 5), avec un second registre

Correction.

```
1   /* Declaration de fonctionnalites supplementaires */
2   #include <stdlib.h> /* EXIT_SUCCESS */
3
4   /* Declaration des constantes et types utilisateurs */
5
6   /* Declaration des fonctions utilisateurs */
7
8   /* Fonction principale */
9   int main()
10  {
11      /* Declaration et initialisation des variables */
12      int x = 4;
13      int y = 5;
14      int z = 3;
15      int aux; /* variable auxilliaire pour la permutation */
16
17      /* copie de la valeur de x dans la variable auxilliaire */
18      aux = x;
19      /* copie de la valeur de y dans x */
20      x = y;
21      /* copie de la valeur de z dans y */
22      y = z;
23      /* copie dans z de l'ancienne valeur de x, depuis la variable auxilliaire */
24      z = aux;
25
26      /* valeur fonction */
27      return EXIT_SUCCESS;
28  }
29
30  /* Definitions des fonctions utilisateurs */
```

Sans le second registre (traduction) :

```
lecture 10 r0
ecriture r0 13
lecture 11 r0
ecriture r0 10
lecture 12 r0
ecriture r0 11
lecture 13 r0
ecriture r0 12
stop
```

Avec le second registre :

```
lecture 10 r1
lecture 11 r0
ecriture r0 10
lecture 12 r0
```

```
ecriture r0 11
ecriture r1 12
stop
```

Correction. *Note aux chargés de TD.*

- *En cours, ils ont vu les variables impératives et le if. Leur sémantique a été donnée par leur traduction en langage amil.*
- *Ils doivent savoir resoudre/reproduire les exos marqués exercices types et faire leur trace sur un exemple quelconque.*
- *On applique la procédure de résolution :*
  - *on se donne des exemples*
  - *on trouve un algorithme en francais*
  - *on traduit l'algorithme en C, en s'aidant de commentaires*
  - *on teste sur les exemples*
- *Le code de la fonction main vide en C a ete présenté rapidement, commenté avec les differents points qu'ils vont voir au semestre. Il est long et peut etre raccourci mais il faut s'assurer qu'ils sachent écrire ce genre de préambule du C avant d'écrire leurs programmes.*
- *Le code leur est toujours donné avec les commentaires, en suivant scrupuleusement l'indentation choisie. Ils doivent bien comprendre que le code et les commentaires sont indissociables. N'hesitez pas a ajouter des commentaires en fonction des difficultés rencontrées dans votre groupe.*

---

3 Exécution conditionnelle d'instructions : *if*

Les programmes suivants réalisent des affichages, pour cela nous utiliserons la fonction `printf`, disponible après avoir inséré `#include <stdio.h>` en début de programme (ligne 3 dans le programme figure 6).

Testez vos programmes sur machine (avec l'aide de votre enseignant).

3.1 Majeur ou mineur ?

Soit la variable `age`, contenant l'âge d'une personne. Écrire un programme qui affiche si cette personne est majeure ou mineure. Indication : `printf("Vous êtes majeur\n");` affiche `Vous êtes majeur !` et un saut de ligne.

Correction.

```
algorithme :
si age >= 18 alors
  affiche majeur
sinon
  affiche mineur
```

```
1   /* Declaration de fonctionnalites supplementaires */
2   #include <stdlib.h> /* EXIT_SUCCESS */
3   #include <stdio.h> /* printf */
4
5   /* Declaration des constantes et types utilisateurs */
6
7   /* Declaration des fonctions utilisateurs */
8
9   /* Fonction principale */
10  int main()
```

```
11  {
12      /* Declaration et initialisation des variables */
13      int age = 16; /* age de la personne */
14
15      if(age >= 18) /* majeur */
16      {
17          /* affiche majeur */
18          printf("Vous êtes majeur.\n");
19      }
20      else /* mineur (age < 18) */
21      {
22          /* affiche mineur */
23          printf("Vous êtes mineur.\n");
24      }
25
26      /* valeur fonction */
27      return EXIT_SUCCESS; /* renvoie OK */
28  }
29
30  /*Definitions des fonctions utilisateurs */
```

3.2 Exercice type : le minimum de 3 valeurs

Soient 3 variables `a`, `b`, `c`, initialisées à des valeurs quelconques. Écrire un programme qui calcule et affiche à l'écran le minimum des 3 valeurs.

Indication : si `x` est une variable contenant l'entier 42, `printf("Solution : %d\n", x);` affichera `Solution : 42` et un saut de ligne.

Correction.

```
algorithme :
soit min = a /* valeur par defaut */
si b < min /* b plus petit que min courant */
  /* b est le min courant */
  min = b
/* min contient min(a,b) */
si c < min /* c plus petit que min courant */
  /* c est le min courant */
  min = c
/* min contient min(min(a,b),c) = min(a,b,c) */
affiche min
```

3.3 Exercice type : Dans une seconde, il sera exactement...

Écrire un programme qui, étant donnée une heure représentée sous la forme de 3 variables : une pour les heures, `h`, une pour les minutes, `m` et une pour les secondes, `s`, affiche l'heure qu'il sera une seconde plus tard. Il faudra envisager tous les cas possibles pour le changement d'heure. Deux exemples de sortie sont :

```
L'heure actuelle est : 23h12m12s
Dans une seconde, il sera exactement : 23h12m13s
```

```
L'heure actuelle est : 23h59m59s
Dans une seconde, il sera exactement : 00h00m00s
```

Pour l'affichage : `printf("L'heure actuelle est : %dh%dm%ds\n", h, m, s);`.

Correction. *Mieux (hors programme) :*
`printf("L'heure actuelle est : %02dh%02dm%02ds\n", h, m, s);`.

Correction.

```
algo :
- affiche l'heure actuelle
- ajoute une seconde
- si tour du cadran des secondes alors
    - remise a 0 des secondes
    - il est une minute de plus
    - si tour du cadran des minutes alors
        - remise a 0 des minutes
        - il est une heure de plus
        - si tour du cadran des heures alors
         - remise a zero des heures
- affiche la nouvelle heure
```

```c
/* Declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* Declaration des constantes et types utilisateurs */

/* Declaration des fonctions utilisateurs */

/* Fonction principale */
int main()
{
    /* soient 23h59m59s */
    int h = 23;
    int m = 59;
    int s = 59;

    /* affiche l'heure actuelle */
    printf("L'heure actuelle est : %dh%dm%ds\n",h,m,s);

    /* une seconde de plus */
    s = s + 1;

    if(s == 60) /* tour du cadran des secondes */
    {
        /* remise a 0 */
        s = 0;

        /* une minute de plus */
        m = m + 1;

        if(m == 60) /* tour du cadran des minutes */
        {
            /* remise a 0 */
            m = 0;
```

```
35
36              /* une heure de plus */
37              h = h + 1;
38
39              if(h == 24) /* tour du cadran des heures */
40              {
41                  /* remise a zero */
42                  h = 0;
43              }
44          }
45      }
46      /* h,m,s contiennent l'heure une seconde plus tard */
47
48      /* affiche l'heure */
49      printf("Dans une seconde, il sera exactement : %dh%dm%ds\n",h,m,s);
50
51      /* valeur fonction */
52      return EXIT_SUCCESS;
53  }
54
55  /* Definitions des fonctions utilisateurs */
56
```
