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

1 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 1 – Un programme C et sa traduction machine

Voici, figure 1, 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 2) 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 1 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.

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 2 – Traces

Le signe égal (=) a un sens bien particulier, il dénote une *affectation*. L'objectif de cette partie du 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 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).

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
   ```

1.1 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 ...» (indication : vous pouvez utiliser plus de deux variables).

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 */
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 ?
```

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

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

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.

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, les étudiants ont vu les variables impératives et le if. Leur sémantique a été donnée par leur traduction en langage amil.*
- *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.*

---

2 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 1).

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

2.1 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 */
```

2.2 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
```

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

Écrire un programme qui, étant donnée une heure représentée sous la forme de trois 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
```

```
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      /* soient 23h59m59s */
13      int h = 23;
14      int m = 59;
15      int s = 59;
16
17      /* affiche l'heure actuelle */
18      printf("L'heure actuelle est : %dh%dm%ds\n",h,m,s);
19
20      /* une seconde de plus */
21      s = s + 1;
22
23      if(s == 60) /* tour du cadran des secondes */
24      {
25          /* remise a 0 */
26          s = 0;
27
28          /* une minute de plus */
29          m = m + 1;
30
31          if(m == 60) /* tour du cadran des minutes */
32          {
33              /* remise a 0 */
34              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
```