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.

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.

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

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).
2. Traduire ce programme C en un programme amil. On supposera que les deux variables sont stockées aux adresses `10` et `11`.
3. (Facultatif). Donner d'autres solutions en assembleur à ce problème (la permutation des contenus des adresses `10` et `11`).
4. (Facultatif). Mêmes questions que précédemment mais pour faire une permutation circulaire de 3 valeurs.

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.

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.

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);`.