Licence 1 - section B
TD 4 d'éléments d'informatique

Catherine RECANATI – Département d'Informatique – Institut Galilée

Semaine du 14 au 18 novembre 2016

1 Conditions de vérité

On rappelle la syntaxe BNF des expressions vue en cours :

```
expr  ::=  constante  |  identificateur  |  (expr)  |  affectation
             |  expr-arith  |  expr-logic  | ...
expr-logic ::= expr op-bin-log expr  | op-un-log expr  | expr op-comp-log expr

op-bin-log ::= &&  |  ||
op-un-log ::= !
op-comp-log ::=  >  |  <  |  >=  |  <=  | ==  |  !=
```

Exercice 1.1 Quelles sont parmi les expressions logiques qui suivent celles qui sont valides ?

1. (x <= 0)
2. (x = 1)
3. ((x < -100) | | (y < 0) & (x = 0))
4. (x == 1)
5. ((x > 0) | | (x < 0))
6. ((x > 0 && (x < 0))
7. (! (x =0))
8. (! (x ==0))
9. ((x < -100) && (y < 0))
10. ! (y)

Correction 1.1 *Quelles sont parmi les expressions logiques qui suivent celles qui sont valides ?*

1. *(x <= 0) est valide*
2. *(x = 1) est valide syntaxiquement, mais sans doute une erreur du programmeur qui a mis un signe d'affectation au lieu d'un signe d'opórateur d'égalité : (x == 1)*
3. *((x < -100) | | (y < 0) & (x = 0)) Ici encore, l'expression est syntaxiquement valide, mais le programmeur a fait la même erreur pour le signe d'affectation, et une autre erreur avec le symbole & (s'il n'est pas redoublé comme dans &&, ce n'est pas l'opérateur logique ET, mais le ET bit à bit*
4. *(x == 1) est valide*
5. *((x > 0) | | (x < 0)) est valide*
6. *((x > 0 && (x < 0)) n'est pas valide (pb de parentheses)*
7. *(! (x =0)) est valide, mais l'affectation a été utilisée par erreur*
8. *(! (x ==0)) est valide*
9. *((x < -100) && (y < 0)) est valide*
10. *! (y) est valide*

Exercice 1.2 Quelle est la valeur de vérité des expressions logiques suivantes (VRAI=non nulle), sachant que la variable entière x a été initialisée à -230, et la variable y à 0 :

1. (x > 0)

2. (x <= 0)
3. (! (x >= 0))
4. (! (y >= 0))
5. (x = 1)
6. (! (x))
7. (x == 1)
8. (y == 0)
9. ((x > 0) | | (x < 0))
10. ((x < 0) && (x > 0))
11. ((x < 0) && ((x > 0) | | (y <= 0)))
12. (((x < 0) && ((x > 0)) | | (y <= 0)))
13. (! (x == 0))
14. (x !=0)
15. ((x < -100) | | (y < 0))
16. ((x < -100) && (y < 0))
17. ! y

Correction 1.2 *Quelle est la valeur de vérité des expressions logiques suivantes :*

1. *(x > 0) FAUX*
2. *(x <= 0) VRAI*
3. *(! (x >= 0)) VRAI*
4. *(! (y >= 0)) FAUX*
5. *(x = 1) VRAI mais il s'agit sans doute d'une affectation malencontreuse*
6. *(! (x)) FAUX*
7. *(x == 1) FAUX*
8. *(y == 0) VRAI*
9. *((x > 0) | | (x < 0)) VRAI*
10. *((x < 0) && (x > 0)) FAUX (quelque soit la valeur de x d'ailleurs)*
11. *((x < 0) && ((x > 0) | | (y <= 0))) VRAI*
12. *(((x < 0) && ((x > 0)) | | (y <= 0))) VRAI*
13. *(! (x == 0)) VRAI*
14. *(x !=0) VRAI*
15. *((x < -100) | | (y < 0)) VRAI*
16. *((x < -100) && (y < 0)) FAUX*
17. *! y VRAI*

2 Instruction conditionnelle if

Exercice 2.1 Quelle heure est-il ?

Écrire un programme principal qui, étant donnée une heure entrée par l'utilisateur sous la forme de 3 variables pour les heures, `h`, les minutes, `m` et les secondes, `s`, affiche l'heure qu'il sera 1 seconde plus tard. Il faudra envisager tous les cas possibles pour le changement d'heure. Deux exemples de sortie sont :

```
Entrez l'heure sous format 24h 12m 56s :  23h 12m 13s (ici entres au clavier)
L'heure actuelle est : 23h12m13s
Dans une seconde, il sera exactement : 23h12m14s

Entrez l'heure sous format 24h 12m 56s :  23h59m59s (ici entres au clavier)
L'heure actuelle est : 23h59m59s
Dans une seconde, il sera exactement : 00h00m00s
```

Correction 2.1 *Quelle heure est-il ?*

Algorithme :

- 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     int h = 0 , m = 0, s = 0;    /* declarations + initialisations */
13     char car;     /* pour lire les separateurs h et m ...   */
14
15     printf(Entrez l'heure sous format 24h 12m 56s: ");
16      scanf("%d%c %d%c %d", &h, &car, &m, &car, &s);
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 */
```

3 Boucle d'instructions while (tant que)

Les boucles sont des blocs d'instructions que l'on répète "en boucle", c'est-à-dire plusieurs fois, et même parfois indéfiniement. Pour que la boucle s'arrête, on peut mettre une condition d'entrée, comme par exemple dans cette boucle `while` (= tant que) dont la syntaxe est la suivante :

```
while (/* Condition d'entree */)
{
    /* bloc d'Instructions a effectuer */
}
```

La condition d'entrée est évaluée avant de chercher à pénétrer dans le bloc d'instructions du `while` pour y exécuter les instructions. Si la condition est fausse, on ignore le bloc d'instructions et on poursuit le programme après le bloc du `while`, comme avec un `if` dont la condition logique est fausse.
Si la condition est vraie, le bloc d'instructions est exécuté une première fois. Puis on recommence (on tente un nouveau tour de boucle), la condition est à nouveau évaluée, et si elle est vraie, le bloc d'instructions est exécuté une fois de plus. Et ainsi de suite, jusque'à ce que la condition d'entrée soit fausse.
Notez que pour que la valeur de vérité de la condition puisse changer, il faut que l'expression de la condition comporte une variable modifiable dans le bloc d'instructions. Ainsi par exemple, la boucle `while` suivante imprimera 5 fois de suite Coucou (avec i valant 0 quand on entre dans le bloc pour la première fois, ensuite il vaudra 1, puis 2, puis 3, puis 4) :

```
int i = 0;
while ( i  < 5)
{
    printf("Coucou");
    i = i + 1;
}
```

Exercice 3.1 Qu'imprime le programme suivant ?

```
5     int main() {
6        int i = 0;
7
8        while ( i  < 5)
9        {
10             printf("i = %d, ", i);
11             i = i + 1;
12       }
13    }
```

Correction 3.1 *Ce programme imprime*

```
i = 0, i = 1, i = 2, i = 3, i = 4,
```

Exercice 3.2 Conversion Fahrenheit-Celsius.

1. Écrire un programme qui affiche la conversion en degrés Celsius des températures de 0 à 10 (exprimées en degrés Fahrenheit). On s'appuiera sur la formule `[C] = ([F] - 32) x 5/9`.
2. Modifier le programme pour qu'on entre au début un entier n quelconque et qu'on affiche ensuite les conversions des tempratures allant de n à n+10 en format float avec 3 chiffres avant la virgule et 2 chiffres après.

Correction 3.2 *Conversion Fahrenheit-Celsius.*

1. *Programme :*

```
 1  /* declaration de fonctions ou constantes externes */
 2  #include <stdlib.h>    /* pour EXIT_SUCCESS */
 3  #include <stdio.h>    /* pour printf et/ou scanf */
 4
 5   /* fonction principale */
 6   int main()
 7   {
 8            /* declaration de variables locales */
 9         float  temp ;          /* pour la temperature */
10        int  i ;
11
12        i = 0;
13        while ( i < 11)
14        {
15            temp = (i - 32)*5/9 ;
16            printf("%d degres F = %f degres C\n", i , temp);
17            i = i + 1 ;
18         }
19         return EXIT_SUCCESS;
20   }
```

2. *Une variante :*

```
 3   int main()
 4   {
 5             /* declaration de variables locales */
 6         float  temp ;          /* pour la conversion de la temperature */
 7         int  i ;
 8         int max ;
 9
10        printf("Entrez une temperature (entiere) en degres F:  ") ;
11        scanf("%d", &i);
12        max = i + 10 ;
13        while ( i < max)
14        {
15            temp = (i - 32)*5/9 ;
16            printf("%d degres F = %3.2f degres C\n", i , temp);
17            i = i + 1 ;
18         }
19         return EXIT_SUCCESS;
20   }
```