IUT Arles– Info 1ère année – Matière AP (Algorithmique)	TD 11 Algorithmique

Exercice I : **Algorithme de conversion entier-binaire.**
On saisit au clavier une suite de CARACTERES terminée par la touche <Entrée>, cette suite de caractères est supposée composée entièrement de chiffres (pas de vérification à faire) et elle est convertie en un nombre binaire (destiné à initialiser une variable entière). Créer un algorithme qui permet de simuler ce fonctionnement.

A titre indicatif, un algorithme de ce type est exécuté lors de l'exécution de la séquence suivante en C++ :

```
                                    int   n;
                                    .
                                    .
                                    .
                                    cin >> n;
Solution
algo vide Conv_Entier_Binaire ()
var

      entier      n;
      caractère   carcour;

début
      n ← 0;
      lire (carcour);
      // on utilise la convention C pour la détection de la
      // touche <Entrée> : '\n'
      tantque (carcour ≠ '\n')
            // la valeur courante de n est multipliée par 10 et
            // augmentée de (carcour - '0') qui correspond à la valeur
            // décimale du caractère tapé
            n ← 10 * n + (carcour - '0');
            // lecture caractère suivant
            lire (carcour);
      fintq;

      afficher (n);

fin;
```

Exemple : l'utilisateur frappe successivement les caractères '1', '2', '3' et <Entrée>.
Codage utilisé : ASCII (cf. table des codes ASCII, facile à obtenir sur Internet)

n ← 0;

carcour ← '1';

n ← 10 * n + (carcour - '0'); // n ← 10*0 + (49 - 48) soit n ← 1

carcour ← '2';

n ← 10 * n + (carcour - '0'); // n ← 10*1 + (50 - 48) soit n ← 12

carcour ← '3';

n ← 10 * n + (carcour - '0'); // n ← 10*12 + (51 - 48) soit n ← 123

carcour ← '\n' // sortie de boucle

Un exemple de solution où la vérification des caractères saisis est faite pour donner une indication de traitement.

```
//=============================================================//
#include <iostream>
using namespace std;
//=============================================================//
int main ()
//-------------------------------------------------------------//
//  ConvEntier02.cpp                                           //
//                                                             //
//  Programme de conversion de caractères en entier avec test  //
//  des caractères saisis                                      //
//  fin saisie = '\n'                                          //
//  Version 2                                                  //
//-------------------------------------------------------------//
{

      char  car;
      int   nbre = 0;

      // saisie des caractères et conversion
      car = cin.get ();
      while (car != '\n')
      {
            // on teste si le caractère lu est un chiffre
            // si ce n'est pas le cas, on ignore ce caractère...
            // choix arbitraire mais acceptable pour 1 tel exemple
            if ((car >= '0') && (car <= '9'))
            {
                  nbre = 10*nbre + (car - '0');
            }
            car = cin.get ();
      }
      // vérification valeur après conversion
      // utilise la conversion inverse : cf. algo conversion binaire-entier
      cout << "\nvaleur apres conversion : " << nbre << endl << endl;

      return 0;

}     // main
//=============================================================//
```

Exercice II : **Algorithme de conversion binaire-entier.**

On suppose qu'une variable interne de type entier doit être affichée à l'écran. Si la valeur interne (elle est en binaire, on raisonne en décimal pour simplifier) est : 123, il faut envoyer à l'écran la suite de CARACTERES '1', '2' et '3' !

A titre indicatif, un algorithme de ce type est exécuté lors de l'exécution de la séquence suivante en C++ :

```
int   n;
.
.
.
cout << n;
```

```
algo vide Conv_Binaire_Entier ()
var

      entier      n,
                  r,
                  i;
      caractère   tableau v[12];

début

      // on part de la valeur mémorisée (en binaire) dans l'entier n
      // pour rappel une opération réalisée sur des nombres en base n
      // donne le même résultat quelle que soit la base
      // si c ← a + b; c aura la même valeur après addition si a et b
      // sont binaires ou si a et b sont décimaux
      // la seule "différence" réside dans le fait que le résultat sera
      // exprimé dans la base des opérandes…
      // on va mémoriser le résultat dans un chaîne de caractères
      // on réserve 12 caractères ce qui est suffisant en architecture
      // 32 bits (valeur maximale : 2^31 - 1)
      i ← 0;
      tantque (n  ≠ 0)
            // on "extrait" le caractère le plus à droite (unités)
            r ← mod (n, 10)             // ou : n % 10
            n ← n / 10;
            // le caractère correspondant aux unités est r + '0'
            // ca caractère est mémorisé en positioni du vecteur
            v[i] ← r + '0';
            // indice du prochain caractère à mémoriser
            i ← i + 1;
            // la nouvelle valeur de n pour extraire le chiffre suivant
            n ← n / 10;
      fintq;

      // ceci pose problème en C/C++ : la solution sera vue en TP
      afficher (v);

fin;
```

Exemple : la variable interne n contient (en binaire !) : 123.
Codage utilisé : ASCII (cf. table des codes ASCII, facile à obtenir sur Internet)
(pour information, le caractère de code 123 en ASCII est : '{')

```
r ← n % 10;                    // r ← 123 % 10 soit 3
n ← n / 10;                    // n ← 123 / 10 soit 12
v[0] ← 3 + '0';                // 3 + 48 soit 51 ou '3'

r ← n % 10;                    // r ← 12 % 10 soit 2
n ← n / 10;                    // n ← 12 / 10 soit 1
v[1] ← 2 + '0';                // 2 + 48 soit 50 ou '2'

r ← n % 10;                    // r ← 1 % 10 soit 1
n ← n / 10;                    // n ← 1 / 10 soit 0
v[2] ← 1 + '0';                // 2 + 48 soit 49 ou '1'
```

Le vecteur v contient la suite de caractères : '3', '2', '1'… elle est inversée par rapport au résultat souhaité : il reste à écrire une fonction d'inversion d'une chaîne (qui heureusement existe dans la bibliothèque)

on peut alors terminer l'algorithme de la façon suivante :

```
// la fonction "inverse" inverse la chaîne de caractères v, le résultat est dans la chaîne origine
// si avant appel, la chaîne contient '3', '2', '1', après exécution elle contient '1', '2', '3'
inverse (v);
afficher (v);
```

REMARQUES :

1/ Il faut se souvenir qu'en interne, un caractère est représenté par son code : il n'y a donc pas '0' mais 48 (encore une fois la valeur binaire de 48 !) ; une opération algorithmique de type caractère + entier qui peut sembler curieuse et en tout cas hétérogène, mais elle ne pose pas de problème au niveau du langage car elle est en réalité entier + entier

2/ on peut transformer ces algorithmes en **fonctions**, des prototypes possibles pourraient être :

```
// reçoit une chaîne de caractères numériques,
// la transforme en entier
entier ConvEntierBinaire (caractère tableau []);
```

(cette fonction en C : atoi, pour comprendre a = chaîne dans la terminologie K&R pour des raisons que j'ai oubliées* - devrait être s car chaîne = string en anglais ! -, i = integer pour K&R, soit "conversion chaîne vers entier")

Nb : K & R : Brian W. Kernighan and Dennis M. Ritchie.

```
// reçoit un entier, le transforme en chaîne de valeur correspondante
vide Conv_Binaire_Entier (caractère tableau [], int);
```

D'autres prototypes seraient envisageables, cf. TP C++.

* = G. REBOULET