Distributeur

Un *distributeur*, de sandwiches par exemple, delivre des produits de différents prix. Il accepte des pièces données ($1, 2, 5$, et 10) et rend éventuellement la monnaie (s'il le peut).

On se propose d'écrire le programme de fonctionnement de cet appareil.

Question 1

Décrire les *objets* nécessaires à la modélisation du distributeur.

Question 2

Écrire un algorithme général montrant les étapes principales du programme, sous forme d'*appels de fonctions* avec leurs arguments et la valeur renvoyée, qui sont évidemment des objets décrits précédemment.

Question 3

Écrire la fonction *Recevoir* les piéces dans le distributeur.

Question 4

Écrire la fonction *calcule* la monnaie : nombre de pièces de chaque catégorie, pour faire une somme donnée.

Question 5

Discuter les *limites de validité* du programme ; y a-t-il des cas extrêmes pour lesquels il ne fonctionne pas correctement ? Peut-on remédier à cela (et comment) ?

```
#define NBP 4

typedef enum {FALSE, TRUE} Boolean ;

void Halte (char * s)
{
    fprint (stderr, " *** %s ! \a\n", s) ;
    exit (1) ;
}

int Initialise (int valP [], int RerV [])
/*
 * Initialise les tableaux :
 *    valP [] : valeurs des pieces
 *    RerV [] : nombre de pieces (de chaque categorie)
 * Renvoie la somme alors contenue
 */
{
}

void Affiche (int valP [], int RsrV [])
/*
 * Affiche le tableau RsrV
 */
{
}


int Recevoir (int valP [], int RsrV [], int prix)
/*
 * Lit (au clavier) les pieces introduites,
 * ignore les pièces non valables,
 * reactualise RsvV [] en consequence,
 * Renvoie la somme totale recue
 */
{
    somme = 0 ;

    while (somme < prix)
    {
        piece = Lire () ;

        for (p = 0 ; p < NBP ; p++)
        {
            if (piece == valP [p])
            {
            somme += piece ;
            RsrV [p] ++ ;
            break ;
        }
     }
    return somme ;
}
```

```
Boolean calcule (int valP [], int RsrV [], int aRendre, int essai)
/*
 * Preleve des pieces dans RsrV
 * en commencant par celle de numero essai
 * jusqu'a concurrence de aRendre
 * Renvoie FALSE si impossible, sinon TRUE.
 */
{
    for (p = essai - 1 ; p >= 0 ; p--)
    {
        v = valP[p] ;

        while (aRendre >= v && RsrV[p] > 0)
        {
            aRendre -= v ;
            RsrV[p] --   ;
        }
    }
    return aRendre == 0 ;
}


main()
{
    Initialise(valP, RsrV) ;

    do
    {
        prix = Lire () ;

        if (prix < 0 ) Halte ("Arret") ;

        TabCpy (OldRsrV, RsrV) ;

        somme = Recevoir(valP, RsrV, prix) ;

        if (somme < 0) Halte ("Arret") ;

        if (somme == 0)
        {
            TabCpy (RsrV, OldRsrV) ; /* Annulation */
            continue ;
        }
        rendre somme - prix ;
        OkMon = calcule (valP, RsrV, rendre, NBP) ;

        if (OkMon) /* delivrer */
        else Halte ("Vide") ,
    } while (1) ;
}
```