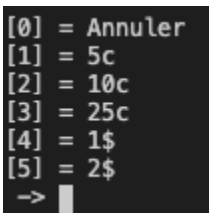
Fonctions obligatoires du projet et le travail demandé :

Sur **Program.cs** :

- Il faut créer un tableau de type Candy : ce tableau servira à charger les données des bonbons dedans à partir du fichier candies.data. cette variable tableau doit être sur le fichier Program.cs. La variable tableau doit être 'private static' : la visibilité private pour quelle soit visible uniquement dans le fichier Program.cs (variable propre à la classe Program) et static pour dire que c'est une variable globale (c'est-à-dire qu'on l'utilisera dans toute les fonction de la classe Program).
- **GetSelection()** : cette fonction doit prendre en paramètres une variable entière qui contiendra le nombre maximale (qui le nombre de bonbons = 25) et qui retourne une selection. Le rôle de cette fonction est de :
  a. Appeler la fonction Print pour l'affichage du bord de la machine à bonbons (l'appel d'une fonction 'static' qui appartient à une classe par : Nom de la classe**.**Nom de la fonction . Exemple : ***Console.WriteLine();*** Console est une classe déjà codée en C# qui contient la fonction *public static string WriteLine())*
  Votre fonction déjà codée Print() est dans la classe Board.
  b. Il faut afficher dans la console la petite flèche qui vient juste après le board et demander à l'utilisateur sa sélection/son choix (la sélection qui doit être entre 1 et le maximum (qui est la variable entière passée en paramètres au départ).
  c. Si l'utilisateur entre une selection qui n'est pas valide (< à 01 ou > à 25) on doit lui redonner la main d'entrer une nouvelle sélection.
  d. Une fois la sélection est valide, on la retourne : *return selection;*

- **GetCandy()** : cette fonction contiendra une seule ligne de code, c'est le return 😊 ! *GetCandy()* prend en paramètres la sélection faite par l'utilisateur (qui est une variable entière entre 1 et 25) et retourne la case du tableau qui contient les données liées à la sélection de l'utilisateur. Autrement dit, si l'utilisateur sélectionne le bonbon numéro 6 par exemple, GetCandy() doit retourner la case du tableau qui contient ce bonbon là qui est dans ce cas candies[5] (le sixième bonbon du tableau est à la position 5 car on commence par la position 0 dans un tableau 😉).

- **GetCoin()** : cette fonction affiche le menu de la monnaie, celui-là dans l'image:



  a. *GetCoin()* donne la main à l'utilisateur d'entrer une valeur de ce menu qui est entre 0 et 5. Si l'utilisateur entre un valeur invalide on lui redonne la main d'entrer une nouvelle valeur…

b. Une fois le choix que l'utilisateur a fait est valide (de 0 à 5), la fonction retourne la valeur de la monnaie qui correspond à son choix (qui est une valeur décimale). exemple : l'utilisateur entre le choix 1, *GetCoin()* retourne la valeur décimale 0.05m (qui vaut 5 sous).
1 (5 sous) => 0.05m , 2 (10 sous) => 0.10m , 3 (25 sous) => 0.25 , 4 (1 dollar) => 1.00m , 5 (2 dollars) => 2.00m

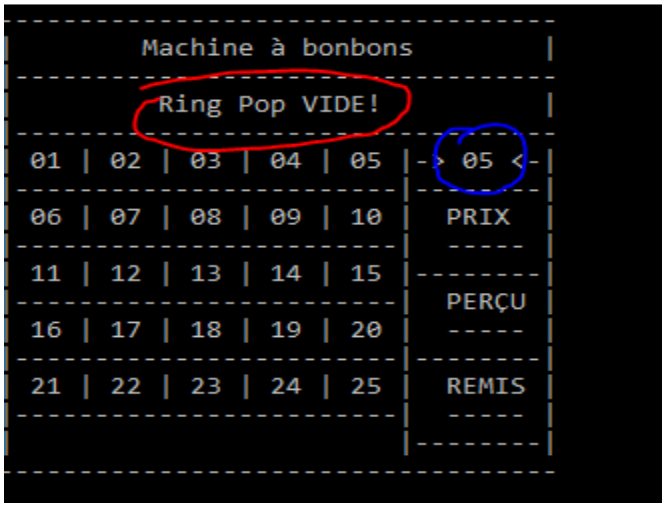
- **LoadCandies()** : une fonction qui permet de charger les données des bonbons du fichier *candies.data* et les mettre dans le tableau des bonbons. C'est une fonction que vous allez l'appeler dans le Main() pour charger toutes les données avant de faire marcher votre machine de bonbons dans la console. *LoadCandies()* est une fonction de la classe Data (qui n'est pas définie avec le mot clé static, c'est-à-dite qu'on ne peut pas l'appeler avec sa classe ~~Data.LoadCandies()~~ mais plutôt en utilisant une variable structurée(objer) de type Data) :

```
Data dataManager = new Data(); /* declaration et reservation de la mémoire de la variable structurée(objet) dataManager de type Data */
candies = dataManager.LoadCandies(); /* appel de la fonction LoadCandies() avec la variable structurée dataManager vu que c'est une fonction propre à la classe Data et qu'on ne peut pas l'utiliser ailleurs sauf en créant une variable de type Data */
```

Ne vous souciez pas, c'est une notion orientée objet que vous allez la voir à la deuxième session 😉

Cette appel de fonction précédemment décrit se fait dans le Main().

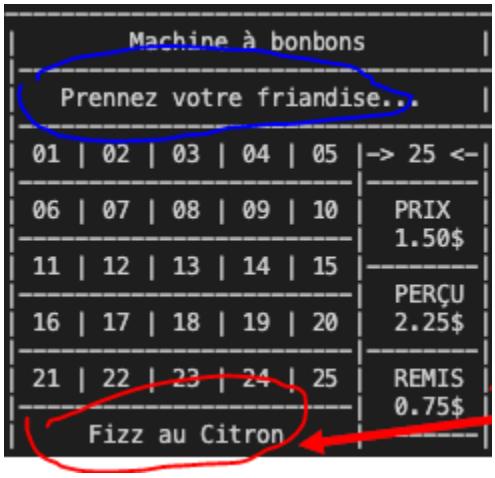
- **La fonction principale Main ()** : tout projet suivra exécute en premier sa fonction principale Main(), donc cette fonction contiendra les instructions de déroulement de votre machine à bonbon :
  a. On récupère la selection de l'utilisateur (*GetSelection())*
  b. On récupère les données du bonbon choisi (*GetCandy())*
  c. On vérifie si le bonbon est en stock. S'il n'est pas en stock on affiche à l'utilisateur sa sélection (entourée en bleu dans la capture qui suit) et le message entouré en rouge 'VIDE!' (*Print())*
  Exemple : l'utilisateur a entré 5, le bonbon Ring Pop correspond au numéro 5 n'est pas en stock => on affiche à l'utilisateur 'Ring Pop VIDE!'



Si le bonbon est en stock, on affiche toujours la selection, le nom du bonbon et son prix.

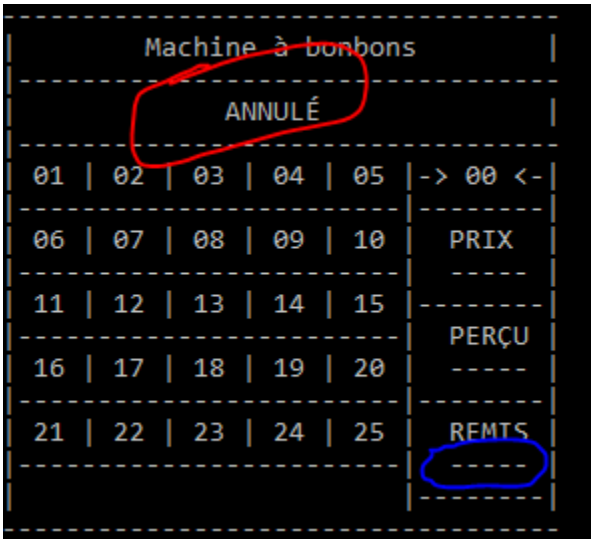
d. On récupère la monnaie entrée par l'utilisateur (*GetCoin()) :*
- Dans le cas oû on a recu de la monnaie de l'utilisateur (ce qu'on a recu de GetCoin() est différent de 0), on l'ajoute à une variable qui contiendra la somme de la monnaie entrée par l'utilisateur et on l'affiche la variable qui contient la somme de la monnaie dans le board (Print()). Exemple : si l'utilisateur a entrée 1.00m (1 dollar) on récupère cette monaie et on l'ajoute dans la variable sommeRecue. Dans le cas ou l'utilisateur aura encore besoin d'entrer plus de monnaire pour compléter le prix demandé du bonbon et il a entrée un autre dollar on va ajouter la nouvelle monnaie à la variable sommeRecue (*sommeRecue = sommeRecue + coin;)*

Si la valeur reçue de la monnaie est plus grande que le prix demandé du bonbon, on calcule le reste qu'on doit retourner à l'utilisateur, on affiche à l'utilisateur la monnaie remise avec le message 'Prenez votre friandise' en haut et le nom du bonbon acheté en bas.(*Print())*



À ce moment là on doit décrémenter notre stock par 1.

- Dans le cas ou on ne trouve pas de monnaie (c'est-à dire GetCoin() est nulle), on affiche le message 'ANNULÉ' à l'utilisateur (Print())



Astuce 1 : utilisez un *do..while* pour gérer la partie d'entrée de la monnaie jusqu'à l'achat (à partir de l'appel de *GetCoin())* . si vous mettez vos instructions (à partir de l'instruction *GetCoin()),* cela vous

permettra de vérifier si l'achat n'est complété ou n'est pas annulé pour relancer les instructions. Pensez à créér deux variable booléennes IsCompleted et IsCanceled que vous utiliserez dans votre code. Do..while vérifera donc si IsCompleted == false et IsCanceled == false pour relancer les instructions et donner la main à l'utilisteur de compléter son achat (ré-entrer la monaie).

Astuce 2 : utilisez une boucle while(true) qui permettra de garder votre machine à bonbon toujours disponible jusqu'à la fermeture de votre console. La boucle rassemble tout le code de la fonction Main() **sauf** la déclaration de la variable structurée dataManager et l'appel de la fonction LoadCandies(). Pensez à ajouter cet affichage à la fin de votre code (dans le while), une fois l'utilisateur appuyera sur un bonton du clavier on lui donnera la main d'utiliser la machine vu que la condition du while est toujours vraie :

```
Console.WriteLine("\nAppuyez sur une touche pour acheter d'autre bonbon...");
```

```
Votre Console.ReadKey() doit être aussi dans le while.
```

Et Voilà c'est tout 😊!