Mon programme se décompose en quatre parties :

* le design,
* les boutons d’event,
* les méthodes de calcul
* et les opérations

# Le design

(-) permet de passer d’un nombre positif à négatif

La flèche ← supprime le nombre ou l’opérande précédent.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| \* | Multiplication | / | Division | - | Soustraction | + | Addition |
| % | Modulo | ^ | Puissance | √ | Racine carrré | ( | Parenthèse G |

# Les boutons d’event

Ils sont au nombre de 6 :

* button1\_Click pour l’ajout des chiffres et des virgules
* button13\_Click pour l’ajout des opérateurs et des parenthèses
* button12\_Click pour le passage en nombre négatif
* button19\_Click pour tout effacer
* button17\_Click pour afficher le résultat en appelant la méthode de calcul egal
* button20\_Click pour supprimer le nombre ou l’opérateur précédant

# Le calcul

Déclaration d’une liste private List<string> calcul = new List<string>(); pour mettre les nombres, les opérateurs et les parenthèses.

Pour respecter les priorités, les appels d’opérations sont placés dans des boucles et on cherche si l’élément de la liste est identique à un des opérateurs dans la boucle.

Mais la véritable boucle intervient dans la récursivité de la méthode RemoveInsert(int i, string total, List<string> calcul). Le but est de supprimer les nombres et l’opérateur qui ont servis et de placer le total à l’endroit du retrait. Ensuite on appelle à nouveau la fonction egal avec cette nouvelle liste.

Les parenthèses utilisent aussi la notion de récursivité pour effectuer le calcul.

# Les opérations

Une classe Operation a été créée pour mieux gérer les opérations et alléger la classe Calcul. Elle regroupe l’ensemble des fonctions destinées à calculer le total de l’opération appelé.

**Projet FilmActeur**

Ce projet est réalisé à l’aide du framework c# .net, il est organisé sous un modèle MVC : Modèles, Vues et Controller. Les vues sont au nombre de trois : SakilaView, FilmView et FilmActorView, tous présent dans le dossier Views>Home. Un layout est aussi présent dans Views>Shared, je l’ai modifié pour appliquer le design bootstrap et fontawesome pour quelques ajouts d’icônes. Une barre de navigation est présente en haut pour naviguer entre les vues.

# Les vues

SakilaView liste tous les acteurs dans l’ordre décroissant et présente un formulaire pour l’ajout d’un acteur. Dans la liste des acteurs, on peut supprimer un acteur.

FilmView est comparable à la vue précédente, sauf qu’on le fait pour les films et pour éviter d’afficher trop de résultat, j’ai limité l’affichage des films aux dix derniers.

FilmActorView permet d’afficher les films qu’à réaliser un acteur avec la possibilité de supprimer la relation (film-acteur) et d’en créer de nouvelles.

# Les modèles

Dans le dossier Models, sont recensés toutes les classes qui interviennent dans le projet avec leur getter et setter, ainsi que leur constructeur. Il y a aussi une classe Builder qui fait le lien avec la base de donnée (ici MySql). J’ai nommé cette classe Sakila, elle permet par exemple de récupérer la liste d’acteur ou de supprimer un acteur à l’aide de l’id en paramètre ou ajouter un acteur en lui donnant un acteur en paramètre.

# Le Controller

Le fait de créer un acteur pour appeler la méthode d’ajout d’un acteur a lieu dans le Controller. On récupère d’abord les variables POST ou GET de la vue que l’on place à la bonne place dans le constructeur de l’objet en question. Puis on insère cet objet dans la méthode associé.

Pour l’affichage des listes, on stocke les données dans des ViewData[] qui sont récupéré par la vue dans les foreach.