**Tutoriel**

**Calculatrice version Native**

**Groupe 1508:**

- ADAMONY Ravel

- MANDAUD Volodia

- ROCHE Hugo

**Enseignant référent:**

- LIMOUZY Vincent

Table des matières

[Introduction 2](#_Toc417288688)

[Mise en place 3](#_Toc417288689)

[Partie graphique 9](#_Toc417288690)

[Partie évènementielle 16](#_Toc417288691)

[Partie calculatoire 22](#_Toc417288692)

[Conclusion 35](#_Toc417288693)

# Introduction

Dans le cadre de notre projet sur *Tizen*, nous avons dans un premier temps rédigé une documentation de présentation sur ce système d’exploitation. Cette documentation comporte une présentation de son architecture ainsi qu’une comparaison face aux géants du marché : *Android* et *iOS*.

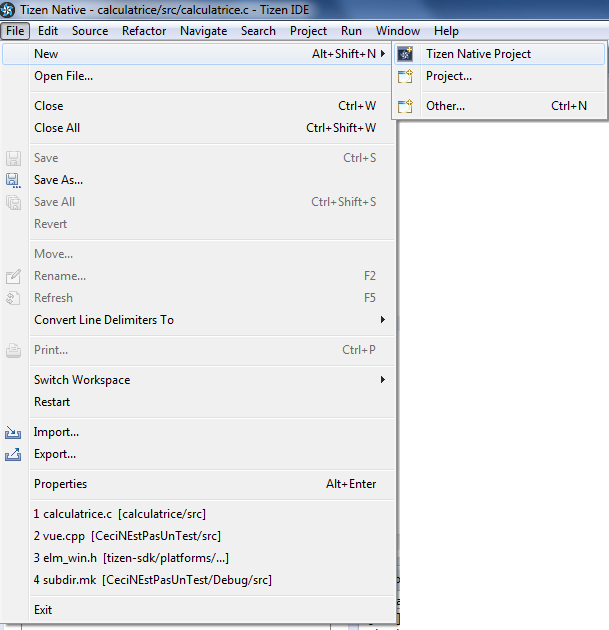
La seconde partie consistait à programmer une application de calculatrice en deux versions distinctes : une version web et une version « native ». Ce choix s’explique par le fait que ces deux types d’applications sont les deux types qui composent *Tizen*. Ce document comporte d’ailleurs une partie où nous comparons les performances des deux applications que nous avons créées.

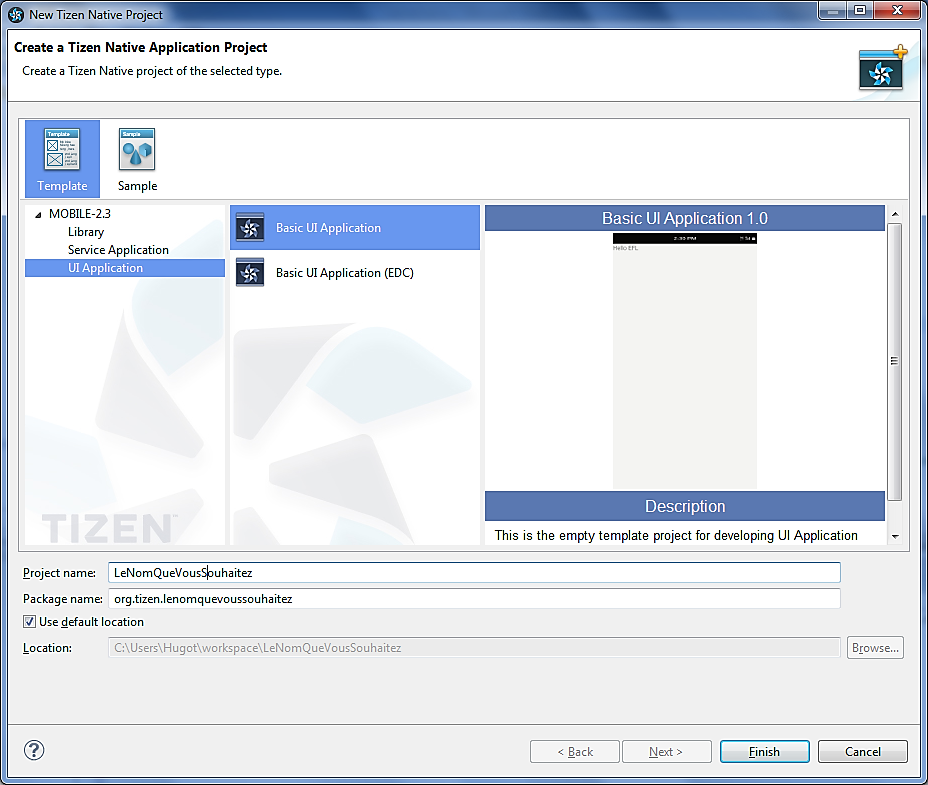
La troisième et dernière partie de notre travail consiste à rédiger un tutoriel expliquant le processus de développement de l’application de calculatrice en version « native ».

Les pré-requis avant d’entamer la lecture de ce document sont d’avoir téléchargé le logiciel [*Eclipse*](https://eclipse.org/downloads/packages/eclipse-ide-java-and-dsl-developers/lunasr2) ainsi que le [*SDK*](https://developer.tizen.org/fr/downloads/tizen-sdk?langredirect=1#2.3) (kit de développement système) de *Tizen*.

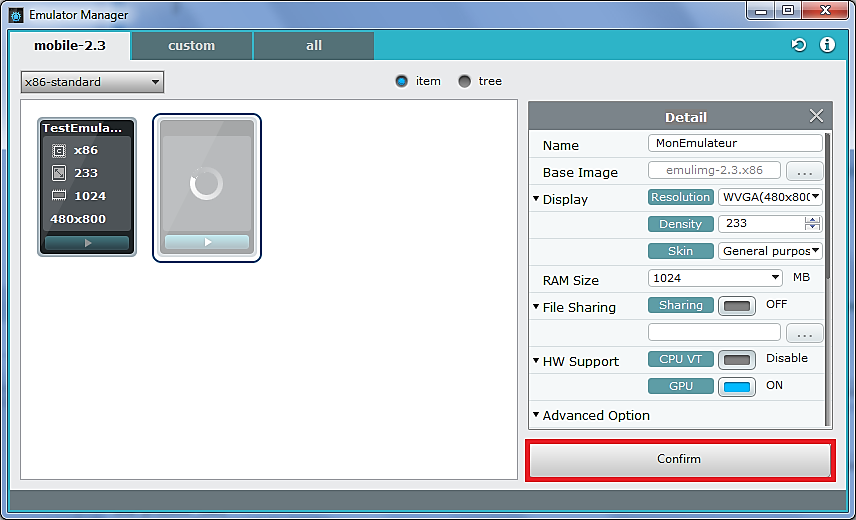
# Mise en place

Lorsque les logiciels évoqués dans l’[introduction](#_Introduction) sont installés correctement, il faut tout d’abord créer un nouveau projet d’application « native » sur le modèle *Simple UI*. Ceci vous donnera une simple fenêtre d’application vierge de laquelle nous pourrons partir.

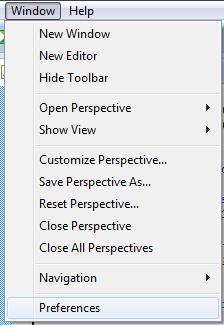
****

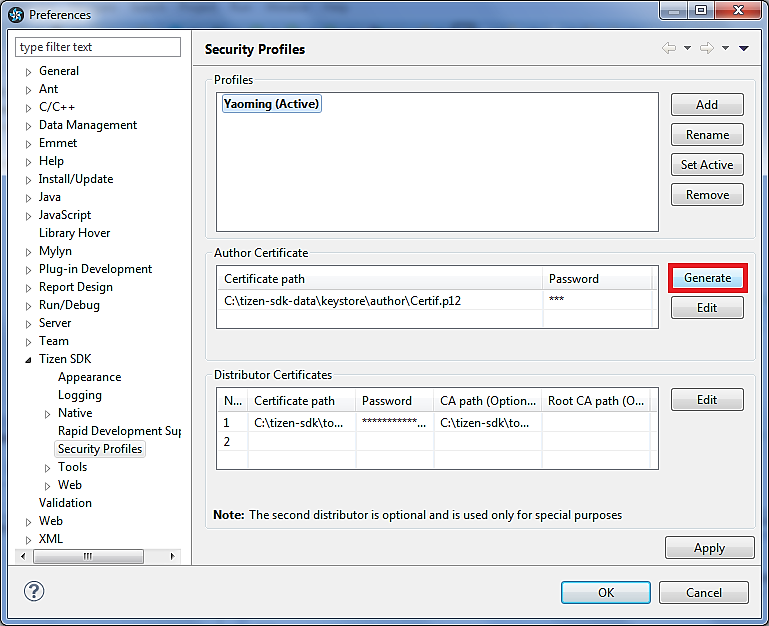


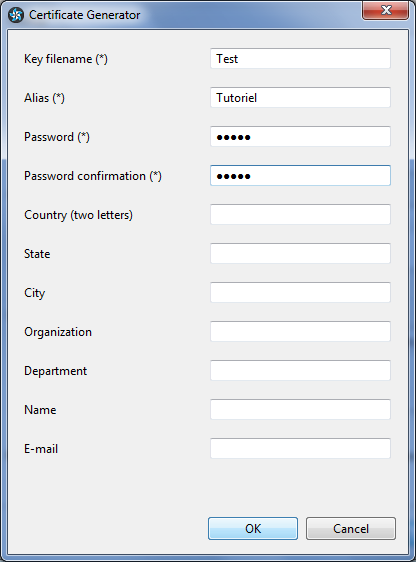
Une fois le projet créé, il faut mettre en place l’émulateur. Pour cela, il suffit de cliquer sur l’*Emulator Manager* fourni par le *SDK* de *Tizen*. Il est possible de garder les paramètres par défaut qui vous sont proposés mais on peut changer la *RAM* attribuée à l’émulateur afin de gagner en vitesse.

****

Une fois votre émulateur créé, il n’est pas encore l’heure de compiler pour vérifier que tout marche ! Il vous faut d’abord générer un nouveau certificat de sécurité. Pour cela, il vous faut accéder à l’onglet préférences.

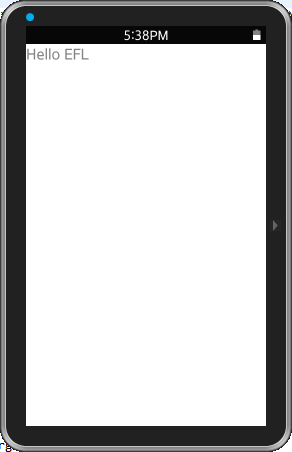






Dans la boîte de dialogue qui s’affiche ensuite, cliquez sur « Yes » ou « Oui » suivant la langue de votre logiciel.

Et voilà, à ce stade si vous compilez vous ne devriez pas avoir de problème et avoir cette fenêtre :



# Partie graphique

Voici un aperçu de l’interface finale:



On distingue donc deux types de boutons : les boutons numéraux et les boutons opérateurs. L’interface est aussi composée de deux zones de texte : une zone de texte se mettant à jour à chaque fois que l’utilisateur clique sur un chiffre et la seconde se met à jour à chaque clic sur un opérateur afin d’afficher l’avancement du calcul.

L’affichage des éléments de l’interface se fait par le biais d’un *grid layout* (affichage sous forme de grille) qui permet d’ordonner les éléments très facilement.

Nous allons utiliser dans cette partie une structure **appdata** (contenant les données de l’application (**appdata** = applicationdata))et la fonction **create\_base\_gui**(permettant de gérer le *GUI* (interface graphique).

Intéressons-nous d’abord aux boutons et à la grille. Comment créer la grille ? Comment y ajouter des éléments ? Rassurez-vous, c’est très simple !

Tout d’abord, il nous faut déclarer un **Evas\_Object** par élément que nous souhaitons ajouter à notre interface. En effet, **Evas\_Object** est le type de base des éléments graphique. Le type d’élément à créer (bouton, zone de texte, etc.) est spécifié lors de l’initialisation de l’objet.

Donc pour créer le bouton 0 et l’ajouter à la grille, il faut d’abord créer les deux **Evas\_Object** correspondant dans la structure **appdata**:

**typedefstruct** appdata {

/\* déclaration des Evas\_Object dans la structure appdata \*/

Evas\_Object \*win; /\* la fenêtre de l’application \*/

Evas\_Object \*conform; /\* ce qui sera affiché \*/

Evas\_Object \*button0;/\* le bouton 0 \*/

Evas\_Object \*grid; /\* la grille \*/

}

Une fois chaque bouton et la grille déclarés comme on vient de le voir, on peut passer à l’initialisation et à l’affichage. Tout cela s’effectue dans la fonction **create\_base\_gui**.

C’est donc ici que nous allons pouvoir spécifier le type de chaque objet que nous avons déclaré avant. Par exemple, pour initialiser notre bouton et la grille il suffit de faire comme suit :

**staticvoid**

**create\_base\_gui**(appdata\_s \*ad)

{

/\*on ajoute un « button » au « conformant » désigné par « conform » qui fait partie de la structure mentionnée auparavant \*/

ad->button0 = **elm\_button\_add**(ad->conform);

//ajout du texte (donc “0”)

elm\_object\_text\_set(ad->button0, "0");

//spécification de la taille souhaitée (20x20 ici)

**evas\_object\_size\_hint\_max\_set**(ad->button0, 20, 20);

}

Et maintenant pour la grille :

/\*mise en place du “Grid Layout” de la même manière que pour les boutons\*/

ad->grid = **elm\_grid\_add**(ad->conform);

/\*ajout du bouton à la grille en spécifiant le positionnement ainsi que les dimensions\*/

//position x: 35, y: 90

//dimension : w: 10, h: 10

**elm\_grid\_pack**(ad->grid, ad->button0, 35, 90, 10, 10);

Si vous compilez à ce stade, ne vous inquiétez pas si rien ne s’affiche : c’est tout à fait normal. Nous n’avons pour l’instant rien affiché ! En effet, nous avons initialisé notre bouton et notre grille, nous avons ajouté le bouton à la grille il faut maintenant ajouter la grille au conformant et afficher le bouton et la grille.

//Ajout de la grille au « conformant »

elm\_object\_content\_set(ad->conform, ad->grid);

//Affichage de tous les éléments

**evas\_object\_show**(ad->button0);

**evas\_object\_show**(ad->grid);

Et voilà ! Mettre en place l’interface graphique n’est pas plus compliqué que ça !

En résumé :

1. Déclaration de l’**Evas\_Object** dans **appdata.**
2. Initialisation et spécification du type d’objet voulu (Cf. documentation *Tizen* pour connaître les fonctions pour chaque type possible).
3. Ajout au layout (ici grid layout) et si on est en train d’initialiser un autre layout, ne pas oublier de l’ajouter au conformant !
4. Affichage de l’objet.

En appliquant ces 4 étapes simples, on arrive facilement à mettre en place l’interface finale.

Voici le code complet d’**appdata** et **create\_base\_gui**:

**staticvoid**

**create\_base\_gui**(appdata\_s \*ad)

{

/\*Les nombres\*/

ad->button0 = **elm\_button\_add**(ad->conform);

ad->button1 = **elm\_button\_add**(ad->conform);

ad->button2 = **elm\_button\_add**(ad->conform);

ad->button3 = **elm\_button\_add**(ad->conform);

ad->button4 = **elm\_button\_add**(ad->conform);

ad->button5 = **elm\_button\_add**(ad->conform);

ad->button6 = **elm\_button\_add**(ad->conform);

ad->button7 = **elm\_button\_add**(ad->conform);

ad->button8 = **elm\_button\_add**(ad->conform);

ad->button9 = **elm\_button\_add**(ad->conform);

elm\_object\_text\_set(ad->button0, "0");

elm\_object\_text\_set(ad->button1, "1");

elm\_object\_text\_set(ad->button2, "2");

elm\_object\_text\_set(ad->button3, "3");

elm\_object\_text\_set(ad->button4, "4");

elm\_object\_text\_set(ad->button5, "5");

elm\_object\_text\_set(ad->button6, "6");

elm\_object\_text\_set(ad->button7, "7");

elm\_object\_text\_set(ad->button8, "8");

elm\_object\_text\_set(ad->button9, "9");

**evas\_object\_size\_hint\_max\_set**(ad->button0, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button1, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button2, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button3, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button4, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button5, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button6, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button7, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button8, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button9, 20, 20);

/\*Operators\*/

ad->buttonAdd= **elm\_button\_add**(ad->conform);

ad->buttonSubs = **elm\_button\_add**(ad->conform);

ad->buttonDiv = **elm\_button\_add**(ad->conform);

ad->buttonMult = **elm\_button\_add**(ad->conform);

ad->buttonResu = **elm\_button\_add**(ad->conform);

ad->buttonC = **elm\_button\_add**(ad->conform);

elm\_object\_text\_set(ad->buttonAdd, "+");

elm\_object\_text\_set(ad->buttonSubs, "-");

elm\_object\_text\_set(ad->buttonDiv, "/");

elm\_object\_text\_set(ad->buttonMult, "\*");

elm\_object\_text\_set(ad->buttonResu, "=");

elm\_object\_text\_set(ad->buttonC, "C");

**evas\_object\_size\_hint\_max\_set**(ad->buttonC, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonAdd, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonSubs, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonDiv, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonMult, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonResu, 40, 40);

//Les zones de texte//

ad->zonecalcul = **elm\_label\_add**(ad->conform);

elm\_object\_text\_set(ad->zonecalcul, "");

**evas\_object\_size\_hint\_weight\_set**(ad->zonecalcul, EVAS\_HINT\_EXPAND, EVAS\_HINT\_EXPAND);

ad->zonesaisie = **elm\_label\_add**(ad->conform);

elm\_object\_text\_set(ad->zonesaisie, "");

**evas\_object\_size\_hint\_weight\_set**(ad->zonesaisie, EVAS\_HINT\_EXPAND, EVAS\_HINT\_EXPAND);

/\*Grid Layout\*/

ad->grid = **elm\_grid\_add**(ad->conform);

**elm\_grid\_pack**(ad->grid, ad->zonecalcul, 5, 5, 90, 20);

**elm\_grid\_pack**(ad->grid, ad->zonesaisie, 20, 25, 90, 20);

//Ajout des nombres//

**elm\_grid\_pack**(ad->grid, ad->button9, 5, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button8, 20, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button7, 35, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button6, 5, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button5, 20, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button4, 35, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button3, 5, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button2, 20, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button1, 35, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button0, 35, 90, 10, 10);

//-----------------------//

//Ajout des opérateurs//

**elm\_grid\_pack**(ad->grid, ad->buttonAdd, 65, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonSubs, 80, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonDiv, 65, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonMult, 80, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonC, 65, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonResu, 80, 75, 10, 10);

//------------------------//

//Ajout de la grille au conformant

elm\_object\_content\_set(ad->conform, ad->grid);

//Affichage de tout le truc (zones, nombres etc...)//

**evas\_object\_show**(ad->button0);

**evas\_object\_show**(ad->button1);

**evas\_object\_show**(ad->button2);

**evas\_object\_show**(ad->button3);

**evas\_object\_show**(ad->button4);

**evas\_object\_show**(ad->button5);

**evas\_object\_show**(ad->button6);

**evas\_object\_show**(ad->button7);

**evas\_object\_show**(ad->button8);

**evas\_object\_show**(ad->button9);

**evas\_object\_show**(ad->buttonC);

**evas\_object\_show**(ad->buttonAdd);

**evas\_object\_show**(ad->buttonSubs);

**evas\_object\_show**(ad->buttonDiv);

**evas\_object\_show**(ad->buttonMult);

**evas\_object\_show**(ad->buttonResu);

**evas\_object\_show**(ad->zonesaisie);

**evas\_object\_show**(ad->zonecalcul);

**evas\_object\_show**(ad->grid);

**typedefstruct** appdata {

Evas\_Object \*win;

Evas\_Object \*conform;

Evas\_Object \*label, \*zonecalcul, \*zonesaisie;

Evas\_Object \*button0, \*button1, \*button2, \*button3, \*button4, \*button5, \*button6, \*button7, \*button8, \*button9;

Evas\_Object \*buttonAdd, \*buttonSubs, \*buttonDiv, \*buttonMult, \*buttonResu, \*buttonC;

Evas\_Object \*grid;

} appdata\_s;

**evas\_object\_size\_hint\_max\_set**(ad->button0, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button1, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button2, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button3, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button4, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button5, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button6, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button7, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button8, 20, 20);

**evas\_object\_size\_hint\_max\_set**(ad->button9, 20, 20);

/\*Les opérateurs\*/

ad->buttonAdd= **elm\_button\_add**(ad->conform);

ad->buttonSubs = **elm\_button\_add**(ad->conform);

ad->buttonDiv = **elm\_button\_add**(ad->conform);

ad->buttonMult = **elm\_button\_add**(ad->conform);

ad->buttonResu = **elm\_button\_add**(ad->conform);

ad->buttonC = **elm\_button\_add**(ad->conform);

elm\_object\_text\_set(ad->buttonAdd, "+");

elm\_object\_text\_set(ad->buttonSubs, "-");

elm\_object\_text\_set(ad->buttonDiv, "/");

elm\_object\_text\_set(ad->buttonMult, "\*");

elm\_object\_text\_set(ad->buttonResu, "=");

elm\_object\_text\_set(ad->buttonC, "C");

**evas\_object\_size\_hint\_max\_set**(ad->buttonC, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonAdd, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonSubs, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonDiv, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonMult, 40, 40);

**evas\_object\_size\_hint\_max\_set**(ad->buttonResu, 40, 40);

//-----------------------------------//

//Les zones de texte//

ad->zonecalcul = **elm\_label\_add**(ad->conform);

elm\_object\_text\_set(ad->zonecalcul, "");

**evas\_object\_size\_hint\_weight\_set**(ad->zonecalcul, EVAS\_HINT\_EXPAND, EVAS\_HINT\_EXPAND);

ad->zonesaisie = **elm\_label\_add**(ad->conform);

elm\_object\_text\_set(ad->zonesaisie, "");

**evas\_object\_size\_hint\_weight\_set**(ad->zonesaisie, EVAS\_HINT\_EXPAND, EVAS\_HINT\_EXPAND);

//-----------------------//

/\*Grid Layout\*/

ad->grid = **elm\_grid\_add**(ad->conform);

**elm\_grid\_pack**(ad->grid, ad->zonecalcul, 5, 5, 90, 20);

**elm\_grid\_pack**(ad->grid, ad->zonesaisie, 20, 25, 90, 20);

//Ajout des nombres//

**elm\_grid\_pack**(ad->grid, ad->button9, 5, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button8, 20, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button7, 35, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button6, 5, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button5, 20, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button4, 35, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button3, 5, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button2, 20, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button1, 35, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button0, 35, 90, 10, 10);

//-----------------------//

//Ajout des opérateurs//

**elm\_grid\_pack**(ad->grid, ad->buttonAdd, 65, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonSubs, 80, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonDiv, 65, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonMult, 80, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonC, 65, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonResu, 80, 75, 10, 10);

//------------------------//

//Ajout de la grille au conformant

elm\_object\_content\_set(ad->conform, ad->grid);

//Affichage de tout le truc (zones, nombres etc...)//

**evas\_object\_show**(ad->button0);

**evas\_object\_show**(ad->button1);

**evas\_object\_show**(ad->button2);

**evas\_object\_show**(ad->button3);

**evas\_object\_show**(ad->button4);

**evas\_object\_show**(ad->button5);

**evas\_object\_show**(ad->button6);

**evas\_object\_show**(ad->button7);

**evas\_object\_show**(ad->button8);

**evas\_object\_show**(ad->button9);

**evas\_object\_show**(ad->buttonC);

**evas\_object\_show**(ad->buttonAdd);

**evas\_object\_show**(ad->buttonSubs);

**evas\_object\_show**(ad->buttonDiv);

**evas\_object\_show**(ad->buttonMult);

**evas\_object\_show**(ad->buttonResu);

**evas\_object\_show**(ad->zonesaisie);

**evas\_object\_show**(ad->zonecalcul);

**evas\_object\_show**(ad->grid);

//Ajout des nombres à la grille//

**elm\_grid\_pack**(ad->grid, ad->button9, 5, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button8, 20, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button7, 35, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button6, 5, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button5, 20, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button4, 35, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button3, 5, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button2, 20, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button1, 35, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->button0, 35, 90, 10, 10);

//-----------------------//

//Ajout des opérateurs à la grille//

**elm\_grid\_pack**(ad->grid, ad->buttonAdd, 65, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonSubs, 80, 45, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonDiv, 65, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonMult, 80, 60, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonC, 65, 75, 10, 10);

**elm\_grid\_pack**(ad->grid, ad->buttonResu, 80, 75, 10, 10);

//------------------------//

//Ajout de la grille au conformant

elm\_object\_content\_set(ad->conform, ad->grid);

//Affichage des objets (boutons, zones de texte et grille)//

**evas\_object\_show**(ad->button0);

**evas\_object\_show**(ad->button1);

**evas\_object\_show**(ad->button2);

**evas\_object\_show**(ad->button3);

**evas\_object\_show**(ad->button4);

**evas\_object\_show**(ad->button5);

**evas\_object\_show**(ad->button6);

**evas\_object\_show**(ad->button7);

**evas\_object\_show**(ad->button8);

**evas\_object\_show**(ad->button9);

**evas\_object\_show**(ad->buttonC);

**evas\_object\_show**(ad->buttonAdd);

**evas\_object\_show**(ad->buttonSubs);

**evas\_object\_show**(ad->buttonDiv);

**evas\_object\_show**(ad->buttonMult);

**evas\_object\_show**(ad->buttonResu);

**evas\_object\_show**(ad->zonesaisie);

**evas\_object\_show**(ad->zonecalcul);

**evas\_object\_show**(ad->grid);

} //fin create\_base\_gui

# Partie évènementielle

Maintenant que nous avons mis en place l’interface nous aimerions qu’il se passe quelque chose lorsqu’on clique sur un bouton ! Dans cette partie nous allons utiliser à la structure **appdata**, la fonction **create\_base\_gui** et une nouvelle fonction que nous appellerons **clicked\_cb** qui sera un callback appelé en cas de clic sur un bouton.

Avant toute chose, il faut associer un évènement à chaque élément de notre *IHM* (Interface Homme Machine). Pour cela il faut rajouter une ligne pour chaque objet dans la fonction **create\_base\_gui :**

**evas\_object\_event\_callback\_add**(ad->button0,*EVAS\_CALLBACK\_MOUSE\_DOWN*, clicked\_cb, ad);

Cette fonction permet d’ajouter à l’objet passé en paramètre un évènement spécifique. Le premier paramètre est donc l’objet sur lequel nous souhaitons ajouter l’évènement (ici le bouton 0). Le second paramètre est une constante désignant le type d’évènement que nous souhaitons gérer, ici le clic (se référer à la documentation *Tizen* pour en avoir la liste exhaustive). Le troisième paramètre désigne la fonction à appeler lorsque l’évènement est déclenché (c’est ici que **clicked\_cb** entre en jeu). Le quatrième paramètre est optionnel et permet de spécifier des données supplémentaires dont nous pourrions avoir besoin. Dans le cas présent, il nous faut avoir accès à l’instance d’**appdata** pour pouvoir modifier les éléments de l’interface.

Une fois cette ligne ajoutée pour chaque bouton, il faut créer la fonction **clicked\_cb**. Avant de s’intéresser à son contenu, étudions sa déclaration (veillez à bien respecter l’ordre des paramètres qui est important):

staticvoid

clicked**\_cb**(**void** \*data, Evas\* e, Evas\_Object \*obj, **void** \*event\_info)

Le premier paramètre va nous permettre de récupérer l’instance d’**appdata** (que nous avons nommé **ad**). Le second paramètre désigne le type d’évènement déclenché (clic, survol, scroll, touche de clavier…). Ce paramètre ne nous intéresse pas puisque nous gérons un seul type d’évènements dans notre cas. Le troisième paramètre désigne l’objet sur lequel l’évènement a été appelé (**ad->button0** par exemple). Le quatrième ne nous intéresse pas.

Intéressons-nous tout d’abord aux évènements des boutons numéraux. Nous voulons mettre à jour l’affichage en fonction du bouton sur lequel nous avons cliqué. Il va donc nous falloir récupérer le texte de la zone de saisie, y ajouter le chiffre correspondant au bouton sur lequel nous avons cliqué et mettre le texte ainsi créé dans la zone de saisie.

appdata\_s \* ad = data;

**constchar**\* temp = elm\_object\_text\_get(ad->zonesaisie); //récupération du texte de la zone de saisie

//Puisque le texte de la zone de saisie est en « const char \* », il faut passer par un « char \* » afin de pouvoir le modifier

**char**\* str;

**strcpy**(str, temp); //récupération du contenu dela zone de texte pour pouvoir le modifier

A ce stade nous avons récupéré le texte de la zone de saisie. Mettons le maintenant à jour.

**char**\* toAppend; //chiffre à rajouter

//gestion de toAppend (en fonction du bouton)

**if** (obj == ad->button0)

{

toAppend = "0";

}

//concaténation de toAppend (donc le nouveau chiffre) à la zone de saisie

**strcat**(str, toAppend);

elm\_object\_text\_set(ad->zonesaisie, str);

Il y aura donc un **if** par bouton afin de pouvoir adapter le comportement aux situations. A ce stade, la zone de saisie devrait se mettre à jour sans problème.

Pour les boutons d’opérateurs, le comportement sera très similaire : pour l’instant on souhaite mettre à jour les deux zones de texte, la partie calculatoire viendra plus tard. On va donc devoir récupérer le texte des deux zones, ajouter l’opérateur sur lequel on a cliqué à la fin de la zone de saisie et ajouter le texte ainsi créé à la fin de la zone de calcul. Voici un exemple pour le bouton « + » par exemple :

//récupération du texte de la zone de calcul

**constchar**\* temp2 = elm\_object\_text\_get(ad->zonecalcul);

**char**\* str2;

**char**\* ope; //opérateur à rajouter

//récupération du contenu dela zone de calcul pour pouvoir le modifier

**strcpy**(str2, temp2);

**if**(obj == ad->buttonAdd)

{

ope = "+";

//on concatène la zone de saisie et l'opérateur et on concatène le tout à la zone de calcul

**strcat**(str, ope);

**strcat**(str2, str);

//réinitialisation du contenu de la zone de saisie et mise à jour de l'affichage de la zone de calcul

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

En généralisant les comportements que l’on vient de voir à tous les boutons, on obtient le code suivant :

**staticvoid**

**clicked\_cb**(**void** \*data, Evas \* e, Evas\_Object \*obj, **void** \*event\_info)

{

appdata\_s \* ad = data;

//récupération texte zone saisie

**constchar**\* temp = elm\_object\_text\_get(ad->zonesaisie);

**char**\* str; //texte zone saisie est en const char\* donc pour pouvoir le modifier il faut passer par un char \*

**constchar**\* temp2 = elm\_object\_text\_get(ad->zonecalcul); //pareil mais pour la zone de calcul

**char**\* str2;

**char**\* toAppend; //chiffre à rajouter

**char**\* ope; //opérateur à rajouter

**if** (ad->calc == 1)

{

elm\_object\_text\_set(ad->zonesaisie, "");

**char** buf[256];

**sprintf**(buf, "%d", ad->res);

elm\_object\_text\_set(ad->zonecalcul, buf);

ad->liste = initialisation();

ajouterEnFin(ad->liste, buf);

ad->calc = 0;

}

//récupération du contenu dela zone de saisie pour pouvoir le modifier

**strcpy**(str, temp);

//récupération du contenu dela zone de calcul pour pouvoir le modifier

**strcpy**(str2, temp2);

//gestion de toAppend (en fonction du bouton)

**if** (obj == ad->button0)

{

toAppend = "0";

}

**elseif**(obj == ad->button1)

{

toAppend = "1";

}

**elseif**(obj == ad->button2)

{

toAppend = "2";

}

**elseif**(obj == ad->button3)

{

toAppend = "3";

}

**elseif**(obj == ad->button4)

{

toAppend = "4";

}

**elseif**(obj == ad->button5)

{

toAppend = "5";

}

**elseif**(obj == ad->button6)

{

toAppend = "6";

}

**elseif**(obj == ad->button7)

{

toAppend = "7";

}

**elseif**(obj == ad->button8)

{

toAppend = "8";

}

**elseif**(obj == ad->button9)

{

toAppend = "9";

}

//concaténation de toAppend (donc le nouveau chiffre) à la zone de saisie

**strcat**(str, toAppend);

elm\_object\_text\_set(ad->zonesaisie, str);

//gestion des opérations

**if**(obj == ad->buttonAdd)

{

//on ajoute à la liste le contenu de la zone de saisie

ajouterEnFin(ad->liste, str);

**if** (ad->parenth == 1)

{

ajouterEnFin(ad->liste, ")");

ad->parenth = 0;

}

ope = "+";

//on ajoute l'opérateur à la liste

ajouterEnFin(ad->liste, ope);

//on concatène la zone de saisie et l'opérateur et le tout on concatène à la zone de calcul

**strcat**(str, ope);

**strcat**(str2, str);

//réinitialisation du contenu de la zone de saisie et de mise à jour de l'affichage de la zone de calcul

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonSubs)

{

ajouterEnFin(ad->liste, str);

**if** (ad->parenth == 1)

{

ajouterEnFin(ad->liste, ")");

ad->parenth = 0;

}

ope = "-";

ajouterEnFin(ad->liste, ope);

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonMult)

{

**if**(ad->parenth != 1)

{

ajouterEnFin(ad->liste, "(");

ad->parenth = 1;

}

ajouterEnFin(ad->liste, str);

ope = "x";

ajouterEnFin(ad->liste, ope);

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonDiv)

{

**if**(ad->parenth != 1)

{

ajouterEnFin(ad->liste, "(");

ad->parenth = 1;

}

ajouterEnFin(ad->liste, str);

ope = "/";

ajouterEnFin(ad->liste, ope);

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->button2)

{

toAppend = "2";

}

**elseif**(obj == ad->button3)

{

toAppend = "3";

}

**elseif**(obj == ad->button4)

{

toAppend = "4";

}

**elseif**(obj == ad->button5)

{

toAppend = "5";

}

**elseif**(obj == ad->button6)

{

toAppend = "6";

}

**elseif**(obj == ad->button7)

{

toAppend = "7";

}

**elseif**(obj == ad->button8)

{

toAppend = "8";

}

**elseif**(obj == ad->button9)

{

toAppend = "9";

}

//concaténation de toAppend (donc le nouveau chiffre) à la zone de saisie

**strcat**(str, toAppend);

elm\_object\_text\_set(ad->zonesaisie, str);

//gestion des opérations

**if**(obj == ad->buttonAdd)

{

ope = "+";

//on concatène la zone de saisie et l'opérateur et on concatène le tout à la zone de calcul

**strcat**(str, ope);

**strcat**(str2, str);

//réinitialisation du contenu de la zone de saisie et mise à jour de l'affichage de la zone de calcul

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonSubs)

{

ope = "-";

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonMult)

{

ope = "x";

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonDiv)

{

ope = "/";

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

A ce stade, les zones de texte devraient se mettre à jour correctement mais les calculs ne sont pas encore effectués.

Le cas de l’opérateur « C » est unpeu spécial, en effet ce bouton nous servira à tout réinitialiser :

**elseif**(obj == ad->buttonC)

{

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, "");

}

} //fin clicked\_cb

# Partie calculatoire

Dans cette partie, nous allons nous intéresser à la structure **appdata** et la fonction **clicked\_cb.** Avant de rentrer dans les détails programmatiques, étudions le fonctionnement que nous souhaitons mettre en place.

Nous allons mettre en place une structure de liste chaînée dans laquelle nous pourrons ajouter les chiffres et les opérateurs et faire le calcul. L’ajout se fera lors du clic sur un opérateur et lors du clic sur le « = » afin d’ajouter la dernière partie du calcul. Nous souhaitons aussi mettre en place la priorité des opérations, il faudra donc ajouter des parenthèses lors des multiplications et divisions.

De façon plus claire, si l’utilisateur clique sur « + » et qu’il a saisi le chiffre « 58 » auparavant dans la zone de saisie, la liste aura cette forme :

|  |  |
| --- | --- |
| 58 |  |

|  |  |
| --- | --- |
| + |  |

Et pour prendre un exemple complet, si on veut faire 58 \*3 + 85 par exemple, la liste aura cette forme :

|  |  |
| --- | --- |
| ( |  |

|  |  |
| --- | --- |
| 58 |  |

|  |  |
| --- | --- |
| \* |  |

|  |  |
| --- | --- |
| + |  |

|  |  |
| --- | --- |
| 85 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| ) |  |

Les parenthèses ont été rajoutées et entoure la multiplication afin de pouvoir gérer la priorité et ainsi effectuer d’abord ce qui est entre parenthèses puis l’insérer dans le reste du calcul. Il n’y aura plus qu’à parcourir la liste pour effectuer le calcul.

Tout d’abord, intéressons-nous à la structure de la liste chainée. Cette structure sera mise en place en deux temps : une structure élément qui contient la valeur et un pointeur vers l’élément suivant et une structure liste qui contient un pointeur vers le premier élément (ce qui nous permettra de parcourir notre structure sans problème). Il nous faudra une fonction d’initialisation de la liste ainsi qu’une fonction pour ajouter les éléments à la fin de la liste.

Voici le code pour cette partie :

//---------------------------------------LA LISTE CHAINEE POUR LES OPERATEURS ET CHIFFRES---------------------------------------//

**typedefstruct** element element;

**struct** element

{

**char**\* val;

element\* nxt;

};

**typedefstruct** Liste Liste;

**struct** Liste

{

element \* premier;

};

Liste \* **initialisation**()

{

Liste \*liste = **malloc**(**sizeof**(Liste));

element\* premElem = **malloc**(**sizeof**(element));

**if** (liste == NULL || premElem == NULL)

{

**exit**(EXIT\_FAILURE);

}

premElem->val = " ";

premElem->nxt = NULL;

liste->premier = premElem;

**return** liste;

}

**voidajouterEnFin**(Liste\* list, **char**\* valeur)

{

//on crée un nouvel élément

element\* nouvelElement = **malloc**(**sizeof**(element));

**char** \* test = **malloc**(**strlen**(valeur));

**strcpy**(test, valeur);

//on assigne la valeur au nouvel élément

nouvelElement->val = test;

//on ajoute en fin, donc aucun élément ne va suivre

nouvelElement->nxt = NULL;

**if**(**strcmp**(list->premier->val, " ") == 0)

{

//si la liste est vide on initialise le premier élément

list->premier = nouvelElement;

}

**else**

{

/\*sinon, on parcourt la liste à l'aide d'un pointeur temporaire et on

indique que le dernier élément de la liste est relié au nouvel élément \*/

element\* temp = list->premier;

**while**(temp->nxt != NULL) //on parcourt jusqu'au dernier élément

{

temp = temp->nxt;

}

//l'élément d'après = nouvelElement

temp->nxt = nouvelElement;

}

}

//------------------------------------------------------------------------------//

**if**(**strcmp**(list->premier->val, " ") == 0)

{

//si la liste est vide on initialise le premier élément

list->premier = nouvelElement;

}

**else**

{

/\*sinon, on parcourt la liste à l'aide d'un pointeur temporaire et on

indique que le dernier élément de la liste est relié au nouvel élément \*/

element\* temp = list->premier;

**while**(temp->nxt != NULL) //on parcourt jusqu'au dernier élément

{

temp = temp->nxt;

}

//l'élément d'après = nouvelElement

temp->nxt = nouvelElement;

}

}

//-------------------FIN LISTE CHAINÉE-------------------//

Maintenant que notre structure est créée, il nous faut en créer une instance dans la structure **appdata** (afin de pouvoir y accéder dans nos évènements).Nous pourrons l’initialiser en faisant appel à la fonction que nous avons codée dans la fonction **create\_base\_gui**.

//à rajouter dans appdata//

**int**calc; //0 si pas de calcul, 1 si calcul effectué (pour rest l'affichage lors du clique sur un chiffre et pas concaténer au résultat)

**double**res; //pour stocker le résultat à chaque calcul

**int**parenth; //0 si pas de parenth, 1 si parenth (pour savoir quand fermer)

Liste\* liste;

//--------------------------------//

//dans create\_base\_gui

ad->liste = initialisation();

ad->calc = 0;

ad->res = 0;

//------------------------------//

Il va maintenant falloir modifier les évènements liés aux boutons d’opérateurs afin de mettre en place l’ajout des éléments à la liste. On souhaite ici gérer la priorité des opérations. Il va donc falloir « encadrer » les multiplications et les divisions par des parenthèses. Pour cela on va rajouter une parenthèse ouvrante lors du clic sur l’opérateur « x » ou « / ». Le booléen présent dans **appdata** (parenth) passera à 1 pour signaler que nous avons ouvert une parenthèse (ainsi on sait qu’il faut la fermer quelque part). Lorsque l’utilisateur clique sur un opérateur autre que « x » ou « / » (donc « + » ou « - » ou « = ») et que parenth est à 1, alors on ferme la parenthèse et la multiplication/division sera bien encadrée. Si toutefois l’utilisateur souhaite faire plusieurs multiplications ou divisions à la suite, les parenthèses ne seront pas fermées. Le code de « x » et « / » devient :

**elseif**(obj == ad->buttonMult)

{

**if**(ad->parenth != 1)//si on a pas ouvert de parenthèse, alors on en ouvre une (->ajout de « ( » à la liste)

{

ajouterEnFin(ad->liste, "(");

ad->parenth = 1;

}

ajouterEnFin(ad->liste, str);

ope = "x";

ajouterEnFin(ad->liste, ope);

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

**elseif**(obj == ad->buttonDiv)

{

**if**(ad->parenth != 1)

{

ajouterEnFin(ad->liste, "(");

ad->parenth = 1;

}

ajouterEnFin(ad->liste, str);

ope = "/";

ajouterEnFin(ad->liste, ope);

**strcat**(str, ope);

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

Et le code du bouton « + » (qui sera le même pour « - » )par exemple devient :

**if**(obj == ad->buttonAdd)

{

//on ajoute à la liste le contenu de la zone de saisie

ajouterEnFin(ad->liste, str);

**if** (ad->parenth == 1)//si on avait ouvert une parenthèse, on la ferme ici (->ajout de « ) » à la liste

{

ajouterEnFin(ad->liste, ")");

ad->parenth = 0;

}

ope = "+";

//on ajoute l'opérateur à la liste

ajouterEnFin(ad->liste, ope);

//on concatène la zone de saisie et l'opérateur et le tout on concatène à la zone de calcul

**strcat**(str, ope);

**strcat**(str2, str);

//réinitialisation du contenu de la zone de saisie et de mise à jour de l'affichage de la zone de calcul

elm\_object\_text\_set(ad->zonesaisie, "");

elm\_object\_text\_set(ad->zonecalcul, str2);

}

Maintenant que cette partie est mise en place, vous devriez arriver au égal avec une liste correcte (à laquelle il manque le dernier élément que nous allons ajouter maintenant). L’opérateur « = » est plus compliqué à expliquer mais nous allons procéder étapes par étapes.

Il va d’abord falloir ajouter le contenu de la zone de saisie afin d’ajouter la dernière partie du calcul à notre liste. Ensuite nous allons initialiser deux valeurs : l’une contiendra le résultat final et la seconde permettra de faire les calculs entre parenthèses.

Pour gérer la priorité des opérations, il va nous falloir ajouter un booléen qui nous permettra de savoir si on a rencontré une parenthèse ouvrante et ainsi adapter notre calcul. En effet, si ce booléen est à « vrai », alors on doit agir sur l’entier permettant de gérer les calculs entre parenthèses et non sur le résultat final.

Un autre booléen nous permettra de savoir si l’on vient d’effectuer un calcul afin de pouvoir réutiliser le résultat et d’empêcher que l’on puisse ajouter des chiffres à la suite du résultat obtenu. Si ce booléen est à « vrai », le résultat est affiché dans la zone de saisie et on ne veut pas que l’utilisateur puisse rajouter des éléments à la suite (car on est dans la zone de saisie dont c’est le rôle principal, il faut donc faire attention), l’affichage des deux zones de texte et la liste sont donc réinitialisés. On ajoute ensuite le résultat à la liste puisqu’il servira de début pour le calcul suivant. Si l’utilisateur ne souhaite pas réutiliser le résultat précédent, il pourra cliquer sur « C » qui réinitialise tout comme on l’a vu.

Étudions le comportement de l’opérateur « = » en ne prenant en compte que l’addition pour commencer :

**elseif**(obj == ad->buttonResu)

{

//on ajoute le dernier élément du calcul à la liste (puisqu'il ne sera ajouté nul part ailleurs)et ajout de « ) » si besoin

ajouterEnFin(ad->liste, str);

**if** (ad->parenth == 1)

{

ajouterEnFin(ad->liste, ")");

ad->parenth = 0;

}

ad->calc = 1;

//on met à jour l'affichage de zone de calcul (pour afficher la fin du calcul)

**strcat**(str2, str);

elm\_object\_text\_set(ad->zonecalcul, str2);

//on récupère le premier élément de la liste

element \* tmp = ad->liste->premier;

//on initialise res comme étant la valeur du premier élément de la liste sauf si la liste commence par une "("

**if** (**strcmp**(tmp->val, "(") != 0)

ad->res = **atoi**(tmp->val);

element \* nextmp;

**int** tempo = 0; //pour l'associativité (le résultat temporaire du truc entre parenthèses)

**int** premierElem = 1; //booléen pour savoir si nous en sommes au premier élément de la liste ou pas

**int** first = 0; //si on a trouvé une parenthèse ouvrante comme 1er élément

**int** parenthese = 0; //si on a trouvé une parenthèse autre part dans le calcul mais pas en première position de la liste

element \* saveOpe;

//on parcours la liste

**while**(tmp->nxt != NULL)

{

nextmp = tmp->nxt;

**if**(**strcmp**(tmp->val, "+") == 0) //si c'est un opérateur (+ par exemple)

{

//on récupère la valeur de tmp (un chiffre donc)

//on récupère la valeur d'après nextmp (autre chiffre) et on les ajoute

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res += **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo += **atoi**(nextmp->val);

}

}

//pour pouvoir sauvegarder l’opérateur qui se trouve avant une parenthèse ouvrante et ainsi pouvoir faire la liaison entre tempo (qui contiendra la valeur du calcul entre parenthèses) et res (le résultat final).

element \* saveOpe;

//on parcours la liste

**while**(tmp->nxt != NULL)

{

nextmp = tmp->nxt;//on récupère l’élément suivant

**if**(**strcmp**(tmp->val, "+") == 0) //si l’élément actuel est un opérateur (+ par exemple)

{

//on récupère la valeur de nextmp (un chiffre ou une parenthèse)

//si nextmp est un chiffre et qu’on avait pas rencontré de parenthèse, alors on met à jour res. Si on avait rencontré une parenthèse et qu’elle n’a pas été fermée encore, on met à jour tempo.

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res += **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo += **atoi**(nextmp->val);

}

}

**if** (nextmp != NULL &&**strcmp**(nextmp->val, "(") == 0) //si n'importe quel élément (le premier exclu) est une parenthèse

{

//on récupère l’élément qui suit la parenthèse (donc un chiffre)

element \* nexnextmp = nextmp->nxt;

//on initialise tempo avec le chiffre qui suit la parenthèse ouvrante

tempo = **atoi**(nexnextmp->val);

saveOpe = tmp; //on sauvegarde l'opérateur (tmp est l’élément qui précède la parenthèse ouvrante)

parenthese = 1;//on spécifie qu’on a trouvé une parenthèse ouvrante

}

**elseif** (premierElem == 1 &&**strcmp**(tmp->val,"(") == 0) //si le premier élément de la liste est une parenthèse

{

//alors on initialise tempo grâce à la valeur de nextmp qui est donc le chiffre qui suit la parenthèse ouvrante

tempo = **atoi**(nextmp->val);

parenthese = 1;//on spécifie qu’on a trouvé une parenthèse ouvrante

first = 1;//on spécifie que le premier élément de la liste est une parenthèse ouvrante

}

**elseif** (**strcmp**(nextmp->val, ")") == 0 && first != 1)//si on trouve une parenthèse fermante et que la parenthèse ouvrante qui s’y rapporte n’était pas le premier élément de la liste

{

parenthese = 0;//on spécifie que la parenthèse a été fermée

//on fait la liaison de tempo avec res suivant la valeur de saveOpe (que nous avions sauvegardé juste avant)

**if** (**strcmp**(saveOpe->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(saveOpe->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(saveOpe->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(saveOpe->val, "/") == 0)

ad->res /= tempo;

}

**elseif** (**strcmp**(nextmp->val, ")") == 0 && first == 1)//si on trouve une parenthèse fermante et que le premier élément de la liste était une parenthèse ouvrante

{

parenthese = 0;//on spécifie que la parenthèse a été fermée

first = 0;//plus besoin de ce booléen, la parenthèse est fermée

//on récupère l’élément qui suit le parenthèse fermante (un opérateur donc)

element \* nexnextmp = nextmp->nxt;

//on récupère l’élément d’encore après (un chiffre)

element \* nexnexnextmp = nexnextmp->nxt;

ad->res = **atoi**(nexnexnextmp->val);

**if** (**strcmp**(nexnextmp->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(nexnextmp->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(nexnextmp->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(nexnextmp->val, "/") == 0)

ad->res /= tempo;

tmp = nexnextmp;

}

**if**(premierElem == 1)

premierElem = 0;

tmp = tmp->nxt;

}

**char** buf[256];

**sprintf**(buf, "%d", ad->res);

elm\_object\_text\_set(ad->zonesaisie, buf);

}

//on initialise res comme étant la valeur du premier chiffre qui suit la parenthèse fermante

ad->res = **atoi**(nexnexnextmp->val);

//on fait la liaison entre tempo et res suivant la valeur de l’opérateur qui suit la parenthèse fermante

**if** (**strcmp**(nexnextmp->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(nexnextmp->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(nexnextmp->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(nexnextmp->val, "/") == 0)

ad->res /= tempo;

tmp = nexnextmp;

}

//nous n’en sommes plus au premier élément, premElem passe à 0

**if**(premierElem == 1)

premierElem = 0;

//on passe à l’élément suivant

tmp = tmp->nxt;

}//fin du while

//création d’un buffer pour la récupération et l’affichage du résultat

**char** buf[256];

//conversion de int vers char grâce à sprintf

**sprintf**(buf, "%d", ad->res);

//mise à jour de l’affichage de la zone de saisie, affichage du résultat

elm\_object\_text\_set(ad->zonesaisie, buf);

}

Maintenant, rajoutons le reste des opérateurs (le code est le même à chaque fois) la boucle while du « = » devient :

//on parcours la liste

**while**(tmp->nxt != NULL)

{

nextmp = tmp->nxt;

**if**(**strcmp**(tmp->val, "+") == 0) //si c'est un opérateur (+ par exemple)

{

//on récupère la valeur de tmp (un chiffre donc)

//on récupère la valeur d'après nextmp (autre chiffre) et on les ajoute

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res += **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo += **atoi**(nextmp->val);

}

}

**elseif**(**strcmp**(tmp->val, "-") == 0)

{

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res -= **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo -= **atoi**(nextmp->val);

}

}

**elseif**(**strcmp**(tmp->val, "/") == 0)

{

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res /= **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo /= **atoi**(nextmp->val);

}

}

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res \*= **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo \*= **atoi**(nextmp->val);

}

}

**if** (nextmp != NULL &&**strcmp**(nextmp->val, "(") == 0) //si n'importe quel élément sauf le premier est une parenthèse

{

element \* nexnextmp = nextmp->nxt;

tempo = **atoi**(nexnextmp->val);

saveOpe = tmp; //on sauvegarde l'opérateur

parenthese = 1;

}

**elseif** (premierElem == 1 &&**strcmp**(tmp->val,"(") == 0) //si le premier élément de la liste est une parenthèse

{

tempo = **atoi**(nextmp->val);

parenthese = 1;

first = 1;

}

**elseif** (**strcmp**(nextmp->val, ")") == 0 && first != 1)

{

parenthese = 0;

**if** (**strcmp**(saveOpe->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(saveOpe->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(saveOpe->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(saveOpe->val, "/") == 0)

ad->res /= tempo;

}

**elseif** (**strcmp**(nextmp->val, ")") == 0 && first == 1)

{

parenthese = 0;

first = 0;

element \* nexnextmp = nextmp->nxt;

element \* nexnexnextmp = nexnextmp->nxt;

ad->res = **atoi**(nexnexnextmp->val);

**if** (**strcmp**(nexnextmp->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(nexnextmp->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(nexnextmp->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(nexnextmp->val, "/") == 0)

ad->res /= tempo;

tmp = nexnextmp;

}

**if**(premierElem == 1)

premierElem = 0;

tmp = tmp->nxt;

}

**elseif**(**strcmp**(tmp->val, "x") == 0)

{

**if**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese != 1)

{

ad->res \*= **atoi**(nextmp->val);

}

**elseif**(nextmp != NULL &&**strcmp**(nextmp->val, "(") != 0 &&**strcmp**(nextmp->val, ")") != 0 && parenthese == 1)

{

tempo \*= **atoi**(nextmp->val);

}

}

**if** (nextmp != NULL &&**strcmp**(nextmp->val, "(") == 0) //si n'importe quel élément sauf le premier est une parenthèse

{

element \* nexnextmp = nextmp->nxt;

tempo = **atoi**(nexnextmp->val);

saveOpe = tmp; //on sauvegarde l'opérateur

parenthese = 1;

}

**elseif** (premierElem == 1 &&**strcmp**(tmp->val,"(") == 0) //si le premier élément de la liste est une parenthèse

{

tempo = **atoi**(nextmp->val);

parenthese = 1;

first = 1;

}

**elseif** (**strcmp**(nextmp->val, ")") == 0 && first != 1)

{

parenthese = 0;

**if** (**strcmp**(saveOpe->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(saveOpe->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(saveOpe->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(saveOpe->val, "/") == 0)

ad->res /= tempo;

}

**elseif** (**strcmp**(nextmp->val, ")") == 0 && first == 1)

{

parenthese = 0;

first = 0;

**if**(nextmp->nxt == NULL) //si on a juste une multiplication

ad->res = tempo;

**if** (**strcmp**(nexnextmp->val, "+") == 0)

ad->res += tempo;

**elseif** (**strcmp**(nexnextmp->val, "-") == 0)

ad->res -= tempo;

**elseif** (**strcmp**(nexnextmp->val, "x") == 0)

ad->res \*= tempo;

**elseif** (**strcmp**(nexnextmp->val, "/") == 0)

ad->res /= tempo;

tmp = nexnextmp;

}

**if**(premierElem == 1)

premierElem = 0;

tmp = tmp->nxt;

}

**else**

{

element \* nexnextmp = nextmp->nxt;

element \* nexnexnextmp = nexnextmp->nxt;

ad->res = **atoi**(nexnexnextmp->val);

**if** (**strcmp**(nexnextmp->val, "+") == 0)

ad->res += tempo;

**else** **if** (**strcmp**(nexnextmp->val, "-") == 0)

ad->res -= tempo;

**else** **if** (**strcmp**(nexnextmp->val, "x") == 0)

ad->res \*= tempo;

**else** **if** (**strcmp**(nexnextmp->val, "/") == 0)

ad->res /= tempo;

tmp = nexnextmp;

}

**if**(premierElem == 1)

premierElem = 0;

tmp = tmp->nxt;

}//fin du while

A ce stade, les calculs devraient marcher sans problème. Il ne nous reste plus qu’à implémenter le comportement après un calcul (réinitialiser l’affichage et la liste avec comme seul élément le résultat du calcul précédent) (prenez garde à rajouter ad->calc = 1; au début du code de l’opérateur « = » ):

**if** (ad->calc == 1)

{

//réinitialisation du texte de la zone de calcul

elm\_object\_text\_set(ad->zonecalcul, "");

//on avait récupéré le contenu de la zone de calcul et l’avait mis dans temp2, on réinitialise donc temp2 ici

temp2 = "";

//création d’un autre buffer pour pouvoir convertir le résultat en chaîne de caractères affichable dans la zone de calcul

**char** tst[256];

sprintf(tst, "%d", ad->res);

//réinitialisation de la liste qui ne contiendra maintenant plus que le résultat du calcul précédent pour pouvoir continuer

ad->liste = initialisation();

ajouterEnFin(ad->liste, tst);

//réinitialisation du booléen

ad->calc = 0;

}

Ceci marque la fin de la partie évènementielle de ce tutoriel. Nous avons essayé d’être le plus clair possible mais ce n’est jamais facile d’expliquer un algorithme et un raisonnement sur papier.

# Conclusion

Et voilà ! Votre calculatrice devrait être fonctionnelle ! Des améliorations peuvent être apportées comme par exemple changer la couleur des boutons, ajouter des fonctionnalités, etc.

Ainsi s’achève donc ce projet sur *Tizen*. Nous avons beaucoup appris et avons pu nous confronter à une technologie que nous ne connaissions pas du tout et que nous avons dû réussir à maîtriser en très peu de temps. Ce projet a donc mis en œuvre nos capacités de recherche documentaire, de rédaction de dossiers professionnels et de mise en place et suivi d’un planning précis. La partie programmation nous a aussi mis à l’épreuve car il fallait réussir à prendre en main *Tizen*, se familiariser avec sa syntaxe, ses fonctions, son *IDE*.