Rapport de maintenance Calculateur Graphique

Analyse Lexicale :

L'analyseur lexical d'expressions mathématiques est un programme écrit en langage C qui permet de décomposer une chaîne de caractères représentant une expression mathématique en une séquence de jetons, facilitant ainsi son traitement ultérieur.

Analyse du code :

Fonction « lire_jeton »:

La fonction parcourt la chaîne de caractères '**s**' et analyse chaque caractère pour identifier différents types de jetons ('**opérateurs**', '**fonctions**', '**réels**', etc.) ensuite stockés dans le tableau '**T**' mis en entré.
Elle utilise des boucles **switch** pour déterminer le type de chaque caractère et **while** pour boucler sur l'entièreté d'un type de caractère.
La fonction traite les cas des **opérateurs**, des **fonctions**, des **variables**, des **réels** et des **erreurs**.
Elle utilise les fonctions **isspace**, **isalpha** et **isdigit** pour vérifier le type de caractère analysé.
Elle utilise aussi **strcmp** pour comparer les chaînes de caractères pour les fonctions prédéfinies.

Fonction « afficher_jeton »:

Cette fonction prend un tableau de jetons et leur nombre en argument, puis les affiche. Elle utilise une boucle for pour parcourir tous les jetons et les afficher en fonction de leur type.

Autres remarques :

Les constantes comme **PAR_OUV**, **PAR_FERM**, etc., sont définies dans le fichier d'en-tête **jeton.h**. La gestion des erreurs est implémentée via un paramètre de pointeur **TypeErreur* erreur**. Cette variable permet de savoir si une erreur est survenue.. Cependant, si plusieurs erreurs se produisent, seule la dernière sera rapportée.

Analyse Syntaxique :

Dans le cadre du développement du calculateur graphique, l'analyse syntaxique joue un rôle essentiel dans le processus de transformation des expressions mathématiques en structures de données compréhensibles par l'ordinateur. L'analyse syntaxique analyse la structure grammaticale de l'expression pour garantir sa cohérence et sa conformité avec les règles définies.

Explication des règles :

Règle « ABS_SIGNE_FOIS » :

L'absence du signe « * » est **autorisé** dans les cas suivants :

- f(x) = 5x
- f(x) = expression(expression)
- f(x) = expression Fonction(Expression)
- f(x) = 5x^3 = (5*x)^3

Elle est **interdite** en cas de factorisation à gauche :

- f(x) = x5
- f(x) = (expression)expression
- f(x) = (expression)x

Voir Annexe 1 et 2.

Règle « OPERATEUR » :

L'utilisation de deux opérateurs de suite, ou d'un opérateur sans expression à droite est interdite.
L'utilisation d'un opérateur sans expression à gauche n'est autorisée que pour « - » et « + ».
Les fonctions suivantes sont donc **autorisées** :

- f(x) = +expression
- f(x) = -expression

Les fonctions suivantes sont donc **interdites** :

- f(x) = 5++5
- f(x) = expression+
- f(x) = *expression

Voir Annexe 3, 4 et 5.

Règle « PARENTHESES » :

On commence toujours par ouvrir une parenthèse avant de la fermer, et il doit y avoir autant de parenthèses ouvertes que fermées.
Les fonctions suivantes sont donc **interdites** :

- f(x) =)5(
- f(x) = (5

Voir Annexe 6 et 7.

Règle « FCT » :

Une fonction s'applique uniquement au terme qui suit et ne contient qu'un unique paramètre, s'il y a un terme « * » muet entre un réel et une variable alors la fonction s'applique à la multiplication de ce réel par la variable :

- f(x) = sin 5 * x = sin (5) * x
- f(x) = sin 5x +1= sin(5x) +1
- f(x) = sin 5 sin cos 6x +1 = sin(5) * sin(cos(6x)) + 1

Voir Annexe 8 à 12.

Fonction « AjouterTermesMuets » :

- Ajout des « * » muets par exemple :
  - sin 5x devient sin(5*x)
  - (expression)(expression) devient (expression)*(expression)
  - 5x (expression) devient (5*x) * (expression)
- Vérification des erreurs de parenthèses, erreurs du type :
  - f(x) = ((sin(x) * 54)
  - f(x) =)5*x(

Création de l'arbre :

- On récupère l'indice à mettre en haut de l'arbre (la dernière opération exécutée)
- Pour récupérer l'indice, on recherche l'index d'un opérateur suivant la priorité choisis.
  - Si l'expression est sous-parenthèses : On trouve la position de la dernière parenthèse et on crée l'arbre associé à l'expression sous parenthèses.
  - Si l'expression n'est pas sous-parenthèses on crée le nœud du jeton
- On crée les fils gauche et droit de manière récursive
- On vérifie que le nœud crée est syntaxiquement correct+

Evaluateur :

Le développement d'une calculatrice graphique requiert la mise en place d'un système robuste pour l'évaluation précise des expressions mathématiques. Le code fourni constitue une composante essentielle de ce système, visant à interpréter et à calculer les expressions entrées par l'utilisateur. Ce rapport offre une analyse détaillée des différentes composantes de ce code.

Analyse du code :

Le code est organisé de manière logique, avec les éléments suivants :

Inclusion de Fichiers d'En-tête :

La première section du code inclut les fichiers d'en-tête nécessaires, tels que "evaluateur.h" et "jeton.h", qui fournissent les déclarations des fonctions et des structures de données utilisées dans le programme.

Fonction d'Évaluation Principale :

La fonction centrale du code est Evaluateur, responsable de l'évaluation des expressions mathématiques. Cette fonction prend en entrée un arbre de jetons et une valeur pour la variable X, et renvoie le résultat de l'expression évaluée.

Traitement des différents types de nœuds :

La fonction Evaluateur gère plusieurs types de nœuds de l'AST, y compris les nœuds représentant des réels, des variables, des opérateurs et des fonctions mathématiques.

Gestion des opérateurs :

Le code prend en charge les opérateurs arithmétiques tels que l'addition, la soustraction, la multiplication, la division et la puissance. Il effectue également des vérifications pour éviter les erreurs de syntaxe, comme la division par zéro.

Gestion des Fonctions Mathématiques :

La fonction Evaluateur supporte plusieurs fonctions mathématiques courantes, telles que sinus, cosinus, tangente, exponentielle, logarithme et partie entière. Ces fonctions sont évaluées en utilisant les fonctions prédéfinies de la bibliothèque mathématique standard.

Fonctionnement de l'Évaluation :

L'évaluation de l'expression est réalisée de manière récursive en parcourant l'arbre de syntaxe abstraite. À chaque nœud rencontré, la fonction Evaluateur évalue son contenu en fonction de son type et des valeurs des nœuds enfants. Cette approche permet une évaluation précise et efficace des expressions mathématiques complexes.

Interface Graphique :

Dans le contexte du développement d'un calculateur graphique, l'interface graphique constitue la façade visible de l'application, offrant aux utilisateurs un moyen intuitif d'interagir avec les fonctionnalités de calcul et de visualisation des données. Elle présente les outils nécessaires pour saisir les expressions mathématiques, ajuster les paramètres de visualisation et interpréter les résultats graphiques obtenus.

Analyse du code :

Fonction « ProcessSpecialKeys » :

Cette fonction prend en paramètre une « key » qui correspond à une touche « spéciale » appuyée (les touches spéciales comprennent les flèches directionnelles, la touche Suppr, Entrer, …). Suivant la touche appuyée (on ne s'intéresse qu'aux flèches directionnelles), on modifie une variable qui permettra dans la fonction « MyDraw » de mettre à jour le graphique.

Fonction « ProcessMouseClick » :

Cette fonction prend en paramètre 4 entiers, correspondant respectivement au bouton de la souris cliquée, à une variable servant à garder en mémoire l'état du clic, et des coordonnées de l'appui. La fonction change la valeur de la variable si le bouton enfoncé et le clic gauche. Si c'est le cas, on garde également en mémoire les coordonnées du clic.

Fonction « graduation » :

Cette fonction permet l'affichage du quadrillage de fond, et des graduations (valeurs et encoches sur les axes). Il y aura toujours un découpage de l'écran en 10 graduations (suivant x et suivant y).

Fonction « affichageFonction » :

Cette fonction permet l'affichage de la courbe de la fonction. Elle prend en paramètre une matrice de taille 2x1001, contenant dans un tableau les valeurs des x et dans l'autre, les valeurs des y = f(x) associées. La fonction va calculer l'abscisse et l'ordonnée de chaque couple de points dans le repère et tracer une ligne droite entre les 2 points calculés.

Fonction « myDraw » :

Cette fonction permet la gestion des actions effectuées par l'utilisateur, l'affichage et la mise à jour de l'écran. Cette fonction commence par remplir l'écran de blanc, puis vérifie si une flèche ou le clic gauche de la souris a été enfoncé. Si c'est le cas, elle effectue les actions associées : se décaler d'un dixième de l'écran dans une direction, dézoomer ou zoomer, mettre en pause l'affichage et demander une nouvelle fonction à l'utilisateur. Si la dernière action a été effectuée, l'utilisateur se verra demander une nouvelle fonction dans le terminal.

Après avoir géré toutes les actions possibles, la fonction recalcule les variables utilisées dans les autres fonction suivant les nouvelles données. Par exemple, si un dézoom a eu lieu, les

valeurs des abscisses et ordonnées minimales et maximales ont changées, le pas doit donc être mis à jour.

Enfin, la fonction myDraw appelle les autres fonctions qui permettent l'affichage, soit « *graduation* » et « *affichageFonction* », tout en retraçant les boutons « Zoom », « Dezoom » et « Changer fonction ».

Fonction « defVariables » :

Cette fonction demande à l'utilisateur de rentrer diverses valeurs permettant de définir ce qui devra être affiché par le calculateur graphique. Les valeurs demandées sont : abscisses et ordonnées minimales et maximales (xmin, xmax, ymin, ymax) ainsi que le nombre de valeurs (permettant donc d'affiner la qualité et la précision de l'affichage. Plus le nombre de valeurs est grand, plus la fonction possèdera de points à afficher et plus elle sera précise). Ce dernier ne doit pas dépasser 1000.

Fonction « main » :

Cette fonction est la fonction exécutée au lancement de l'exécutable. Elle appelle tout d'abord la fonction defVariables afin de permettre à l'utilisateur de choisir les valeurs importantes. Elle les affiche ensuite dans le terminal et lance le module d'affichage grâce à la fonction « *InitGraph* » qui appellera alors la fonction « myDraw » a chaque action de l'utilisateur.

Annexes (Analyse Syntaxique)

Annexe 1 :

TEST 1: f(x) = x(5x+1)
Objectif : Vérifier que l'analyse syntaxique gère le cas de l'absence du signe « * »
Cas particuliers traités :
1) 5x → 5*x
2) x(5*x+1) → x*(5*x+1)
Erreur attendue : AUCUNE

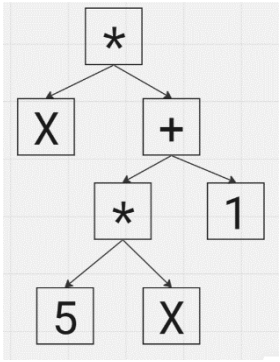

Fig 1 : Arbre attendu (TEST 1)

```
TEST 1: Objectif : Verifier que l analyse syntaxique gere le cas de l'absence du signe  *
Cas particuliers traites :
1) 5x -> 5*x
2) x(5*x+1) -> x*(5*x+1)
Erreur attendue : AUCUNE

 Fonction: f(x) = x(5x+1)

Tableau recu:
[X, PAR_OUV, 5, X, +, 1, PAR_FERM, FIN]

Apres Traitement erreurs parentheses:
[X, *, PAR_OUV, 5, *, X, +, 1, PAR_FERM, FIN]

Arbre renvoye:
  *
|->|->  X
|->|->  +
|->|->|->|->  *
|->|->|->|->|->|->  5
|->|->|->|->|->|->  X
|->|->|->|->  1
```

Fig 2 : Résultat (TEST 1)

Annexe 2 :

TEST 2 : f(x) = (x+5)5
Objectif : Vérifier que l'analyse syntaxique gère le cas de l'absence du signe « * »
Erreur attendue : ERREUR_ABS_SIGNE_FOIS

```
TEST 2:  f(x) = (x+5)5
Objectif : Verifier que l analyse syntaxique gere le cas de l absence du signe *
Erreur attendue : ERREUR_ABS_SIGNE_FOIS

Tableau recu:
[PAR_OUV, 5, X, +, 1, PAR_FERM, X, FIN]

Apres Traitement erreurs parentheses:
[PAR_OUV, 5, *, X, +, 1, PAR_FERM, X, FIN]

 Erreur: ERREUR_ABS_SIGNE_FOIS
```

Fig 3 : Résultat (TEST 2)

Annexe 3 :

TEST 3 : f(x) = x++x

Objectif : Vérifier que l'analyse syntaxique gère le cas d'un doublon d'opérateurs
Erreur attendue : ERREUR_OPERATEUR

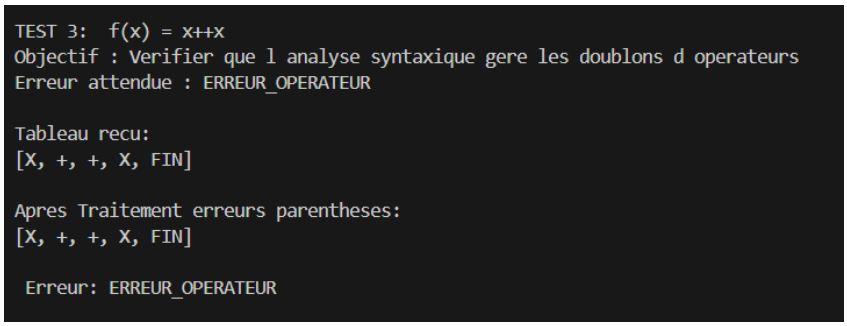

Fig 4 : Résultat (TEST 3)

Annexe 4 :

TEST 4 : f(x) = *x

Objectif : Vérifier que l'analyse syntaxique gère le cas d'un opérateur * ou ^ sans expression à sa gauche.
Erreur attendue : ERREUR_OPERATEUR

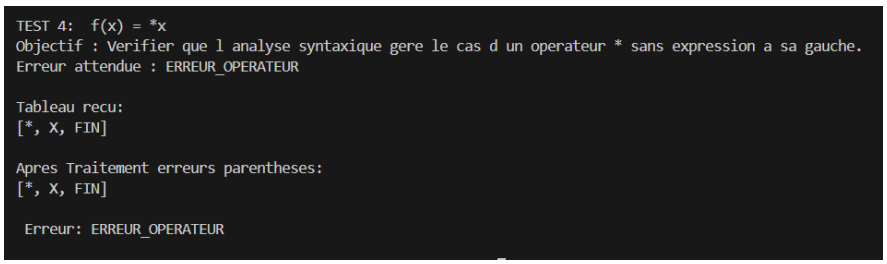

Fig 5 : Résultat (TEST 4)

Annexe 5 :

TEST 5 : f(x) = -(x+1) +1

Objectif : Vérifier que l'analyse syntaxique gère le cas d'un opérateur – sans expression à gauche de cet opérateur.

```
TEST 5 :  f(x) = -(x+1)+1
 Objectif : Verifier que l analyse syntaxique gere le cas d un operateur - sans expression a gauche.
Erreur attendue : AUCUNE

Tableau recu:
[-, PAR_OUV, X, +, 1, PAR_FERM, +, 1, FIN]

Apres Traitement erreurs parentheses:
[-, PAR_OUV, X, +, 1, PAR_FERM, +, 1, FIN]

Arbre renvoye:
  +
|->|->  -
|->|->|->|->  +
|->|->|->|->|->|->  X
|->|->|->|->|->|->  1
|->|->  1

 Erreur: AUCUNE
```

Fig 6 : Résultat (TEST 5)

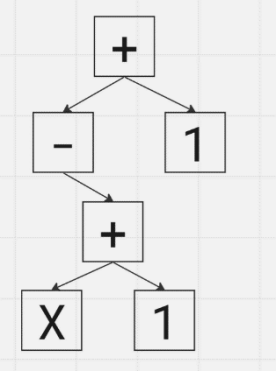

Fig 7 : Arbre Attendu (TEST 5)

Annexe 6 :

TEST 6 : f(x) =)5(

Objectif : Vérifier que l'analyse syntaxique gère les erreurs de parenthèses
Erreur attendue : ERREUR_PARENTHESES

```
TEST 6:  f(x) = )5(
Objectif : Verifier que l analyse syntaxique gere les erreurs liees aux parentheses.
Erreur attendue : ERREUR_PARENTHESES

Tableau recu:
[PAR_FERM, 5, PAR_OUV, FIN]

Apres Traitement erreurs parentheses:
[PAR_FERM, 5, *, PAR_OUV, FIN]

 Erreur: ERREUR_PARENTHESES
```

Fig 8 : Résultat (TEST 6)

Annexe 7 :

TEST 7 : f(x) = (5

Objectif : Vérifier que l'analyse syntaxique gère les erreurs de parenthèses
Erreur attendue : ERREUR_PARENTHESES

```
TEST 7:  f(x) = (5
Objectif : Verifier que l analyse syntaxique gere les erreurs liees aux parentheses.
Erreur attendue : ERREUR_PARENTHESES

Tableau recu:
[PAR_OUV, 5, FIN]

Apres Traitement erreurs parentheses:
[PAR_OUV, 5, FIN]

 Erreur: ERREUR_PARENTHESES
```

Fig 9 : Résultat (TEST 7)

Annexe 8 :

TEST 8 : f(x) = sin cos sqrt 3 *6

Objectif : Vérifier que l'analyse syntaxique gère l'absence de parenthèses pour les fonctions

Erreur attendue : AUCUNE

```
TEST 8:  f(x) = sin cos sqrt 3 * 6
Objectif : Verifier que l analyse syntaxique gere les erreurs liees aux fonctions.
Erreur attendue : AUCUNE

Tableau recu:
[SIN, COS, SQRT, 3, *, 6, FIN]

Apres Traitement erreurs parentheses:
[SIN, COS, SQRT, 3, *, 6, FIN]

Arbre renvoye:
  *
|->|->  SIN
|->|->|->|->  COS
|->|->|->|->|->|->  SQRT
|->|->|->|->|->|->|->|->  3
|->|->  6

 Erreur: AUCUNE
```

Fig 10 : Résultat (TEST 8)

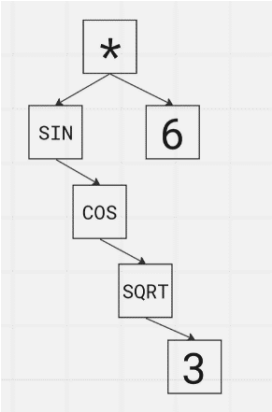

Fig 11 : Arbre attendu (TEST

Annexe 9 :

TEST 9 : f(x) = sin(cos 6 + sqrt 3) +1

Objectif : Vérifier que l'analyse syntaxique gère l'absence de parenthèses pour les fonctions

Erreur attendue : AUCUNE

```
TEST 9:  f(x) = sin (cos 6 + sqrt 3)+1
Objectif : Verifier que l analyse syntaxique gere les erreurs liees aux fonctions.
Erreur attendue : AUCUNE

Tableau recu:
[SIN, PAR_OUV, COS, 6, +, SQRT, 3, PAR_FERM, +, 1, FIN]

Apres Traitement erreurs parentheses:
[SIN, PAR_OUV, COS, 6, +, SQRT, 3, PAR_FERM, +, 1, FIN]

Arbre renvoye:
  +
|->|->  SIN
|->|->|->|->  +
|->|->|->|->|->|->  COS
|->|->|->|->|->|->|->|->  6
|->|->|->|->|->|->  SQRT
|->|->|->|->|->|->|->|->  3
|->|->  1

 Erreur: AUCUNE
```

Fig 12 : Résultat (TEST 9)

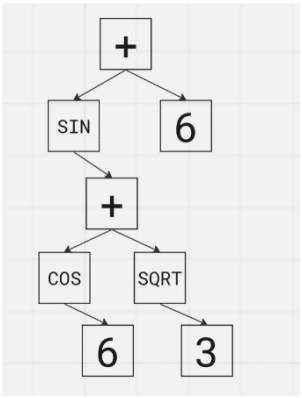

Fig 13 : Arbre attendu (TEST

Annexe 10 :

TEST 10 : f(x) = 6x sin 5 x
Objectif : Vérifier que l'analyse syntaxique gère l'absence de parenthèses et des « * » muets pour les fonctions
Erreur attendue : AUCUNE

```
TEST 10: Objectif : Verifier que l analyse syntaxique gere le cas de l'absence du signe et de parenth¿ses dans le cas de fonctions
Cas d un signe fois fantome pour une fonction
Erreur attendue : AUCUNE

 Fonction: f(x) = 1 + 6x sin 5x



Tableau recu:
[1, +, 6, X, SIN, 5, X, FIN]

Apres Traitement erreurs parentheses:
[1, +, PAR_OUV, 6, *, X, PAR_FERM, *, SIN, PAR_OUV, 5, *, X, PAR_FERM, FIN]

Arbre renvoye:
  +
|->|->  1
|->|->  *
|->|->|->|->  *
|->|->|->|->|->|->  6
|->|->|->|->|->|->  X
|->|->|->|->  SIN
|->|->|->|->|->|->  *
|->|->|->|->|->|->|->  5
|->|->|->|->|->|->|->  X

 Erreur: AUCUNE
```

Fig 14 : Résultat (TEST 10)

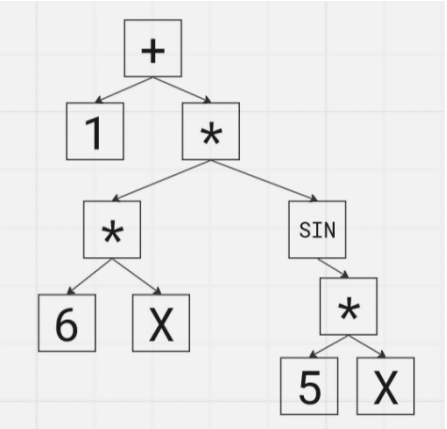

Fig 15 : Arbre Attendu (TEST

Annexe 11 :

TEST 11 : f(x) = sin 5 x cos 6 x + 1
Objectif : Vérifier que l'analyse syntaxique gère l'absence de parenthèses et des « * » muets pour les fonctions
Erreur attendue : AUCUNE

```
TEST 11: Objectif : Verifier que l analyse syntaxique gere le cas de l'absence du signe dans le cas de fonctions
Erreur attendue : AUCUNE

 Fonction: f(x) = sin 5x cos 6x + 1



Tableau recu:
[SIN, 5, X, COS, SIN, 6, X, +, 1, FIN]

Apres Traitement erreurs parentheses:
[SIN, PAR_OUV, 5, *, X, PAR_FERM, *, COS, SIN, PAR_OUV, 6, *, X, PAR_FERM, +, 1, FIN]

Arbre renvoye:
  +
|->|->  *
|->|->|->  SIN
|->|->|->|->|->  *
|->|->|->|->|->|->|->  5
|->|->|->|->|->|->|->  X
|->|->|->|->  COS
|->|->|->|->|->|->  SIN
|->|->|->|->|->|->|->  *
|->|->|->|->|->|->|->|->|->  6
|->|->|->|->|->|->|->|->|->  X
|->|->  1

 Erreur: AUCUNE
```

Fig 16 : Résultat (TEST 11)

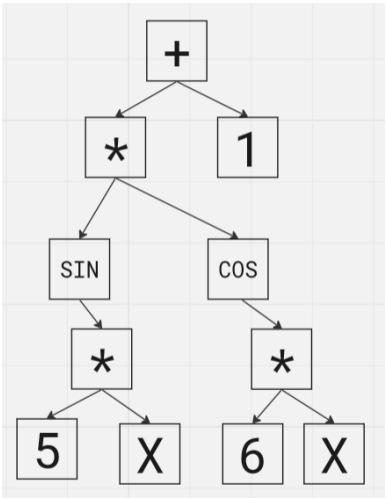

Fig 17 : Résultat (TEST 11)

Annexe 12 :

TEST 12 : f(x) = sin ()
Objectif : Vérifier que l'analyse syntaxique gère le cas d'une fonction sans paramètre
Erreur attendue : ERREUR_FCT_SANS_PARAM

```
TEST 12: Objectif : Tester une fonction sans parametre
Erreur attendue : ERREUR_FCT_SANS_PARAM

 Fonction: f(x) = sin ()



Tableau recu:
[SIN, PAR_OUV, PAR_FERM, FIN]

Apres Traitement erreurs parentheses:
[SIN, PAR_OUV, PAR_FERM, FIN]

Arbre renvoye:
  SIN

 Erreur: ERREUR_FCT_SANS_PARAM
```

Fig 18 : Résultat (TEST 12)

Annexe 13 :

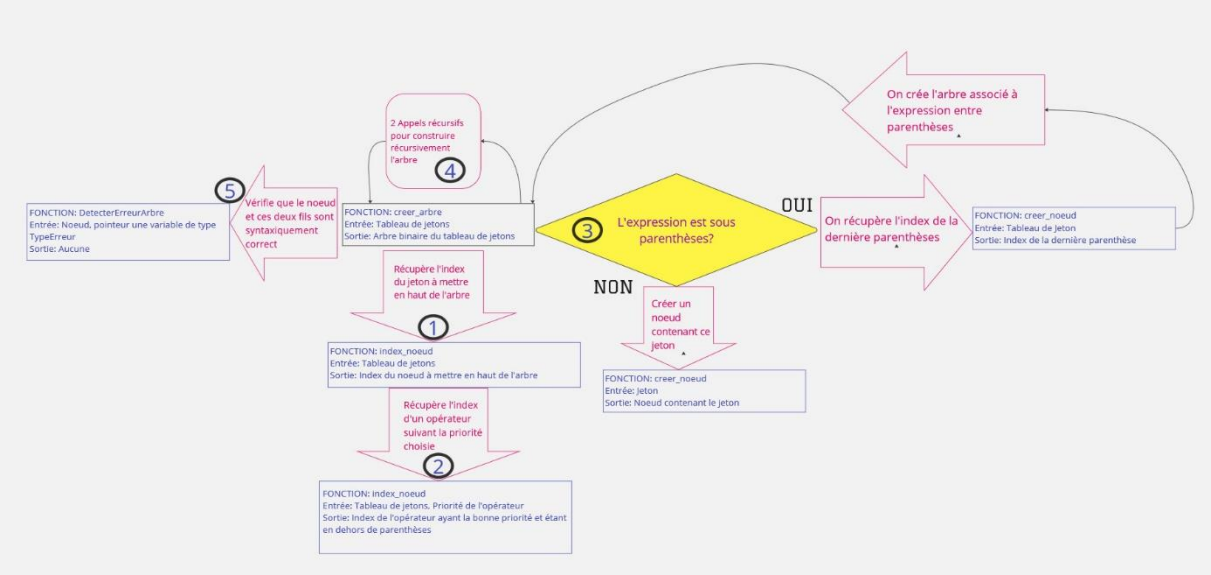

Fig 17 : Principe de fonctionnement de la création d'un arbre