# **ANALYSE**

Pour une analyse approfondie et une résolution efficace du problème en question, une compréhension approfondie des expressions arithmétiques s'avère essentielle. Voici quelques caractéristiques importantes à connaître :

* Une expression arithmétique est composée d'opérandes et d'opérateurs.
* Les opérations suivent l'ordre de priorité des opérateurs :
* '\*' et '/' ont la même priorité.
* '+' et '-' ont la même priorité.
* '\*' et '/' ont une priorité supérieure à '+' et '-'.
* '+' et '\*' sont des opérateurs commutatifs.
* '-' et '/' ne sont pas commutatifs ; le calcul s'effectue de gauche à droite.
* La division par zéro est impossible.
* Une expression est considérée invalide si elle contient des caractères qui ne sont ni des opérandes ni des opérateurs.
* Un opérande peut débuter par un '-' ou un '+' (opérateurs unaires).
* Un opérande n'est pas valide s'il contient plus d'un point ou un caractère non numérique.

Pour l’expression arithmétique il va être saisie au clavier sur un seul ligne donc on récupère les operateurs et les opérandes un par un et on créer pour chaque éléments un nœud et on le saisir dans la bonne place dans notre arbre selon plusieurs condition (la priorité, la commutativité,…. ).

**Choix des variables :**

Pour le choix des variables on a besoin d’un type de donner qui nous permet de saisir soit un réel ou un caractère c’est pour cela on utilise le type union qui nous donne cette possibilité.

**Algorithme de construction de l’arbre :**

Concernant la construction de l'arbre :

Si le nœud contenant l'opérande est vide, il devient la racine de l'arbre. Sinon, il est inséré comme fils du dernier opérateur.

Si le nœud contenant l'opérateur est vide et que l'arbre est vide, il devient la racine, et le nœud contenant l'opérande est son fils gauche. Sinon, il s'insère selon les règles suivantes :

Pour un opérateur '+', l'arbre peut devenir un de ses fils gauche ou droit, selon une variable booléenne indiquant la direction d'insertion, et l'opérateur devient la racine.

Pour un opérateur '-', l'arbre devient son fils gauche (la soustraction se fait de gauche à droite), et l'opérateur devient la racine.

Pour un opérateur '\*' ou '/', et si le dernier opérateur inséré est '+' ou '-', l'un des fils de la racine devient le fils gauche de l'opérateur, selon la direction de la dernière insertion, et l'opérateur devient ce fils.

Pour un opérateur '' ou '/', et si le dernier opérateur inséré est '' ou '/', le dernier opérateur inséré devient le fils gauche de l'opérateur. Si l'arbre contient déjà le dernier opérateur inséré, l'opérateur devient la racine, sinon il devient le fils droit de la racine.

En ce qui concerne l'évaluation de l'expression arithmétique, elle doit suivre un parcours infixé de l'arbre, en utilisant le fils gauche comme premier opérande, la racine comme opérateur, et le fils droit comme deuxième opérande.

**Exemple de la construction de l’arbre pour cette expression arithmétique :** 3\*-1+6/4-1

Récupération de :

1. opérateur :\* ; opérande :3

racine courant

\*

3

NB : le pointeur courant pointe sur le dernier operateur saisie

1. opérateur :+ ; opérande :-1

racine courant

+

\*

3 -1

1. opérateur:/ ; opérande :6

racine

+ courant

\* /

3 -1 6

1. opérateur:- ; opérande :4

racine courant

-

+

\* /

3 -1 6 4

1. opérande :1

racine courant

-

+ 1

\* /

3 -1 6 4

**Evaluation :**

**E =** 3\*-1+6/4-1

**E =** (-3) + 1.5 – 1

**E =** (-3) + 1.5 – 1

**E =** (-4) + 1.5

**E =** -2.5

# **ANALYSE FONCTIONNELLE**

1. Fonction principale « **Expression\_Arithmetique\_Arbre** » : elle permet de lire une expression arithmétique et de la représenter sous forme d’un arbre et de l’évaluer.

1-a) Spécification des données :

Une expression arithmétique est un ensemble d’opérandes et d’opérateurs qui sera représentée sous forme d’un arbre binaire.

1-b) Spécification des résultats :

Le résultat est un arbre contenant une expression arithmétique, et le résultat de son évaluation.

1. Fonction Principale « **Expression\_Arithmetique\_Pile** » : elle permet de lire une expression arithmétique et de la représenter sous forme d’une pile et de l’évaluer.

2-a) Spécification des données :

Une expression arithmétique est un ensemble d’opérandes et d’opérateurs qui sera représentée sous forme d’une pile.

2-b) Spécification des résultats :

Le résultat est une pile contenant une expression arithmétique, et le résultat de son évaluation.

Pour effectuer la construction d’un arbre ou d’une pile, nous définissons les fonctions suivantes qui serons communs :

* Fonction « **Transforme**» : permet d’extraire un opérande et un opérateur de la chaine de caractères lue.
* Fonction « **EpeurerBlanc** » : permet d’ignorer les espaces entre les opérandes et les opérateurs.
* Fonction « **EstNumerique** » : permet de vérifier si un caractère est un chiffre.
* Fonction « **EstOperateur** » : permet de vérifier si un caractère est un opérateur arithmétique.

1. Fonction « **Construire\_Arbre\_Exp** » : elle permet de lire une expression arithmétique et de construire un arbre binaire qui représente cette expression.

3-a) Spécification des données et des résultats :

Cette fonction retourne un arbre binaire qui représente une expression arithmétique.

3-b) Spécification fonctionnelle :

* Fonction « **CreerNoeud** » : permet d’allouer un emplacement mémoire à un nœud.

# **DOSSIER DE PROGRAMMATION**

L’objectif du programme et de représenter une expression arithmétique sous un arbre binaire et de l’évaluer.

1. Type des données :

Une expression est un ensemble d’opérandes et d’opérateurs.

Le résultat est arbre binaire ou une pile contient une expression arithmétique, puis le résultat de son évaluation.

Puisque on a les opérandes sont des nombres réels et les opérateurs sont des caractères, et on veut construire avec ses deux variables un arbre binaire dont chaque nœud contient un opérande ou un opérateur, alors on va utiliser l’union pour regrouper ses deux variables.

Pour construire l’arbre binaire, on va définir une structure qui contient 3 champs : un pour stoker un opérande ou un opérateur, deux champs qui pointent, un sur le fils gauche et l’autre sur le fils droit.

Pour construire la pile, on va définir une structure qui contient 2 champs : un pour stoker un opérande ou un opérateur et un pointeur qui pointe sur la cellule suivante.

1. Librairies :

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

1. Constantes :

#define RC '\n'

#define BL ' '

#define PL '+'

#define MN '-'

#define ML '\*'

#define DV '/'

1. Variables principales :

typedef union // Definition d'une union

{

    float operande;

    char operateur;

}Exp;

typedef struct ND // Definition de la structure d'arbre

{

    Exp info;   // champ d'information de l'arbre

    struct ND \*fgc, \*fdt; // pointeur de fils gauche et droit

}Noeud;

1. Programmation des fonctions :

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : CreerNoeud;

    Sorties     : Noeud \*NE : le noeud cree

    Description : la fonction permet de creer un noeud et de le retourner.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

Noeud \*CreerNoeud()

{

    Noeud \*NE;

    NE = (Noeud \*)malloc(sizeof(Noeud));    // Allocation de la memoire

    if (!NE)

    {

        printf("\nErreur d'allocation de la memoire!");

        exit(0);

    }

    NE->fgc = NE->fdt = NULL;   // Initialiser les fils au NULL

    return ((Nœud\*)NE) ;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : Epeurer\_Blanc;

    Entrees     : char c : un caractere

    Sorties     : char c : un caractere non blanc

    Description : la fonction permet d'ignorer les blancs jusqu'a trouver un

                caractere non blanc et le retourne.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

char Epeurer\_Blanc(char c)

{

    return ((char)(c == BL) ? Epeurer\_Blanc(getchar()):c);

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : EstNumerique;

    Entrees     : char c : un caractere

    Sorties     : int : entier pour indiquer si le caractere est un chiffre

    Description : la fonction permet d'indiquer si un caractere est un

                chiffre s'elle retourne un entier entre 0 et 9, un blanc

                s'elle retourne 11, un point s'elle retourne 12, autre si -1.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int EstNumerique(char c)

{

    if(('0'<=c) && (c<='9')) return ((int)(c-'0')); // le caractere numerique

    if(c == ' ') return ((int)11); // le caractere est un blanc

    if(c == '.') return ((int)12); // le caractere est un point

    return ((int)(-1));

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : EstOperateur;

    Entrees     : char c : un caractere

    Sorties     : int : entier pour indiquer si le caractere est un operateur

    Description : la fonction permet d'indiquer si un caractere est un

                operateur arithmetique, elle retourne 1 si '+', 2 si '-', 3

                si '\*', 4 si '/', 0 si un retour chariot et -1 si autre.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int EstOperateur(char c)

{

switch (c)

    { case RC : return ((int)0);

        case PL : return ((int)1);

        case MN : return ((int)2);

        case ML : return ((int)3);

        case DV : return ((int)4);

        default : return ((int)-1);

    }

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : Transforme;

    Entrees     : float \*operande : pour stoker la valeur de nombre extrait

    Sorties     : char c : le caractere non blanc apres le nombre extrait

    Description : la fonction permet d'extrait un nombre reel dans une chaine

                de caracteres saisie et retourne le caractere non blanc apres

                le nombre extrait.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

char Transforme(float \*operande)

{

    int control; // pour stocker la valeur retournee par "EstNumerique(char)"

    int firstPass = 1; // pour tester la premiere lecture

    int f = 0; // pour distinguer la partie entiere et la partie reel

    int signe = 1; // pour stocker le signe du nombre

    float result = 0; // pour stocker le nombre extrait

    char c = getchar();

    c = Epeurer\_Blanc(c); // Epeurer blanc

    do

    {

        if(firstPass == 1) // Lors de la lecture du premier caractere

        {

            firstPass = 0; // pour indiquer qu'on a lu le premier caractere

            if ((c == '-') || (c == '+'))

            {   // recuperer le signe du nombre

                if (c == '-') signe = -1;

                c = getchar();

                continue; // passer a l'iteration suivante

            }

        }

        control = EstNumerique(c);

        switch (control)

        {

            case -1:

                printf("\nErreur: caractere non numerique!");

                exit(0);

            case 11: // un blanc

                c = Epeurer\_Blanc(c);

                \*operande = (float)(result\*signe);

                return ((char)c);

case 12: // un point

                if(f == 0) // on y encore dans la partie entiere

                {

                    f = -1; // passer a la partier reel

                    break;

                }

                printf("\nErreur: un autre point est tape!");

                exit(0);

            default: // caractere numerique

                if (f == 0) result = result\*10 + (float)control;

                else result += (float)((float)(control)\*pow(10,f--));

        }

        c = getchar();

    } while ((c != PL) && (c != MN) && (c != ML) && (c != DV) && (c != RC));

    \*operande = (float)(result\*signe);

    return ((char)c);

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : Construire\_Arbre\_Exp;

    Sorties     : Noeud \*Arb : un arbre d'une expression arithmetique

    Description : la fonction permet de lire une expression arithmetique et

                de la representee sous forme d'un arbre binaire.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

Noeud \*Construire\_Arbre\_Exp()

{

    Noeud \*Arb = NULL, \*NOperateur, \*NOperande, \*courant;

    char operateur; // pour stocker l'operateur lue

    float operande; // pour stocker l'operande extrait

    int control; // pour stocker la valeur retournee par "EstOperateur(char)"

    int dir = 1; // pour indiquer la direction d'insertion d'un sous arbre

    printf("\nEntrer une expression arithmetique : ");

    while ((operateur = Transforme(&operande)) != RC)

    {

        control = EstOperateur(operateur);

        if (control == -1)

        {

            printf("\nOperateur non defini!");

            exit(0);

        }

        NOperande = CreerNoeud(); // Creer un noeud pour l'operande

        NOperande->info.operande = operande; // affecte la valeur de l'operande

        NOperateur = CreerNoeud(); // Creer un noeud pour l'operateur

        NOperateur->info.operateur = operateur; // affecter le l'operateur

        if (!Arb) // Arbre vide

        {

            Arb = NOperateur; // l'operateur devient racine de l'arbre

            Arb->fgc = NOperande; // l'operande devient son fils gauche

            courant = Arb; // initialiser courant par la racine de l'arbre

        }

        else

        {   // l'operande devient le fils libre de dernier operateur inserer

            if (!courant->fdt) courant->fdt = NOperande;

            else courant->fgc = NOperande;

            // si l'operateur est un '+' ou '-'

            if ((control == 1) || (control == 2))

            {

                if (control == 1) // le cas d'un '+'

                {

                    if (dir == 0) NOperateur->fgc = Arb;

                    else NOperateur->fdt = Arb;

                    dir = 1-dir; // changer dir pour la prochaine insertion

                }

                // cas d'un '-' l'arbre devient sous arbre gauche de l'operateur

                else NOperateur->fgc = Arb;

                Arb = NOperateur; // l'operateur devient la racine de l'arbre

            }

            else // l'operateur est '\*' ou '/'

            {   // le dernier operateur inserer est '+'

                if (courant->info.operateur == PL)

                {

                    if (dir == 0)

                    {

                        NOperateur->fgc = Arb->fgc;

                        Arb->fgc = NOperateur;

                    }

                    else

                    {

                        NOperateur->fgc = Arb->fdt;

                        Arb->fdt = NOperateur;

                    }

                    dir = 1 - dir; // changer dir pour la prochaine insertion

                }

                else

                {   // le dernier operateur inserer est '-'

                    if (courant->info.operateur == MN)

                    {

                        NOperateur->fgc = Arb->fdt;

                        Arb->fdt = NOperateur;

                    }

                    else // le dernier operateur inserer est '\*' ou '/'

                    {   // le dernier operateur devient fils gauche

                        NOperateur->fgc = courant;

                        if (Arb == courant) Arb = NOperateur;

                        else Arb->fdt = NOperateur;

                    }

                }

            }

            courant = NOperateur;

        }

    }

    NOperande = CreerNoeud(); // Creer un noeud pour l'operande

    NOperande->info.operande = operande; // affecte la valeur de l'operande

    if (!Arb) return ((Noeud \*)NOperande); // si l'arbre est vide

    if (!courant->fdt) courant->fdt = NOperande;

    else courant->fgc = NOperande;

    return ((Noeud \*)Arb);

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : Calculer\_Exp;

    Entrees     : float op1      : le premier operande

                  char operateur : un operateur

                  float op2      : le deuxieme operande

    Sorties     : float result   : le resultat de l'operation

    Description : la fonction permet d'effectuer une operation arithmetique

                de deux operandes et de retourner le resultat.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

float Calculer\_Exp(float Op1, char operateur, float Op2)

{

    switch (operateur)

    {

        case '+': return ((float)Op1 + Op2);

        case '-': return ((float)Op1 - Op2);

        case '\*': return ((float)Op1 \* Op2);

        case '/':

            if (Op2 != 0) return ((float)Op1 / Op2);

            // Cas de division par 0

            printf("\nErreur: Division par 0!!");

            exit(0);

    }

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    Nom         : Evaluer\_Arbre\_Exp;

    Entrees     : Noeud \*Arb : la racine de l'arbre

    Sorties     : float      : le resultat d'evaluation de l'expression

                arithmetique

    Description : la fonction permet d'evaluer l'expression arithmetique qui

                est represente sous forme d'un arbre binaire en effectuant un

                parcour infixe de l'arbre.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

float Evaluer\_Arbre\_Exp(Noeud \*Arb)

{

    if (Arb)

    {

        if (!Arb->fgc && !Arb->fdt) return ((float)Arb->info.operande);

        return ((float)Calculer\_Exp(Evaluer\_Arbre\_Exp(Arb->fgc),

                                    Arb->info.operateur,

                                    Evaluer\_Arbre\_Exp(Arb->fdt)));

    }

}