



Compte Rendu

Résolution d'expression arithmétique

Filière Ingénieur :

**Ingénierie Logicielle et
Intégration des Systèmes
Informatiques**

Réalisé par :

El mehdi salah BEN SOUDA

Mohamed JEBBANEMA

Encadré par :

Prof. Abdelkrim BEKKHOUCHA

2022/2023

Table de matières

Chapitre1 : Résolution d'expression arithmétique à l'aide d'un arbre	3
I. Analyse :	3
II. Analyse fonctionnelle	4
III. Dossier de programmation :	7
Chapitre2 : Résolution d'expression arithmétique à l'aide de pile	21
I. Analyse :	21
II. Analyse fonctionnelle	22
III. Dossier de programmation :	28

Chapitre1 : Résolution d'expression arithmétique à l'aide d'un arbre

I. Analyse :

Problème :

Résoudre une expression arithmétique à l'aide d'un arbre binaire.

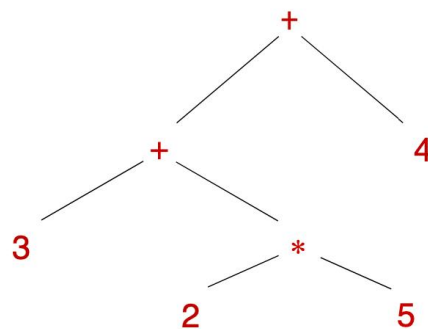


Figure 1: $3+2*5+4=17$

Extraction des données à exploiter :

- L'utilisateur va saisir une chaîne de caractère.
- La lecture de l'expression se fait caractère par caractère.
- Dans une expression saisie, il faut distinguer entre les opérations et les opérandes
- Les opérations possèdent un ordre par rapport à leur priorité :
 - (+, -) ont la même priorité entre eux ;
 - (*, /) ont la même priorité entre eux ;
 - (*, /) sont prioritaires sur (+, -), si ces derniers sont binaires ;
 - (+, -) s'elles sont unaires (début de l'expression), elles sont considérées comme étant un signe ;
 - Pour les opérations (*, /) le calcul doit se faire dans le sens gauche->droite ;
- La validation d'une expression :

- L'expression n'est pas valide s'il contient des caractères qui ne sont ni opérande ni opération.
- Si l'expression saisie commence par une opération (*, /), ça sera non valide ;
- On ne peut pas diviser par 0 ;
- Un opérande ne peut contenir qu'une seule virgule s'il est décimal ;
- (+, -) ne peuvent pas être suivies par (* ou /) ;

II. Analyse fonctionnelle

- 1- **Fonction "est_opérateur"** : la fonction test si le caractère est un opérateur mathématique
 - **Spécifications des données et des résultats :**
On passe un caractère comme donnée, et la fonction retourne 1(le caractère est un opérateur) 0 si non.
- 2- **Fonction "Est_numerique"** : la fonction test si le caractère est un chiffre.
 - **Spécifications des données et des résultats :**
Entrée, un caractère. Si c'est un chiffre, on le retourne, sinon on retourne un entier négatif.
- 3- **Fonction "Characters_To_float"** : Elle extrait un réel (un opérande) d'une chaîne de caractère.
 - **Spécification des données et des résultats :**
 - ~ Une chaîne de caractères.
 - ~ Le résultat est un réel signé (un opérande), le caractère (opérateur) qui suit l'opérande {+, *, /, -, \n} et un message d'erreur, le cas de '\n' c'est le cas où le réel n'est pas suivi par un opérateur.

EXEMPLE

les chaînes de caractères numériques valides :

1.3 , 2- , 3/ , 4* , -5.

les chaînes de caractères numériques qui ne sont pas valides :

3.3a3 , 1a, -a12 , *2 , /3 .

4- **Fonction "priorite"** :elle détermine l'opérateur le plus prioritaire c'est a dire le premier opérateur qu'il faut l'évaluer .

- **Spécification des données et des résultats :**

- ~ Deux caractères.
- ~ Le résultat est un entier(1 ou 0).

L'opérateur 1 (celui de gauche)	L'opérateur 2 (celui de droit)	Résultat
Moins prioritaire	Plus prioritaire	1 (opérateur 2)
Plus prioritaire	Moins prioritaire	0 (opérateur 1)
Meme priorité	Meme priorité	0 (gauche->droite)

5- **Fonction " rendre_ancetre"** : Elle insère un operateur a la bonne place dans un arbre selon les priorités des autres opérateurs .

- **Spécification des données et des résultats :**

Un arbre et l'opérateur a insérer.

Le résultat est un nouveau arbre et l'emplacement de l'insertion

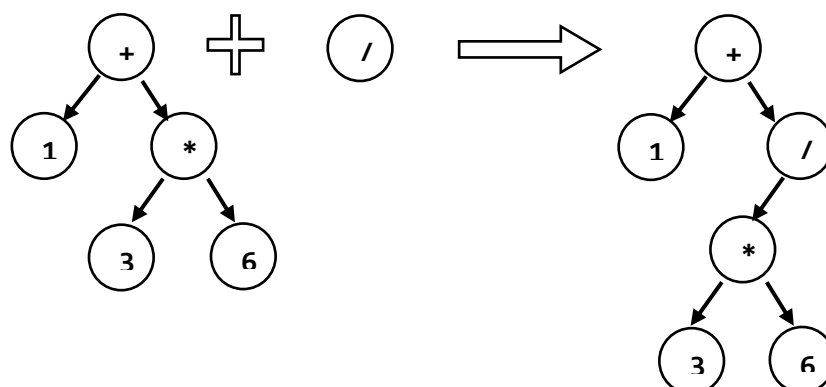
- **Spécification fonctionnelle:**

A chaque fois on compare operateur par opérateur jusqu'à la trouver l'emplacement ou il faut insérer l'opérateur passer en paramètre.

Les feuilles sont des opérande et tous le reste des nœud sont des opérateurs

La comparaison s'effectue entre l'opérateur passer en paramètre et les operateurs des fils gauche.

Après avoir trouver l'emplacement, le sous arbre de ce dernière sera le fils gauche d'un nouveau nœud qui contient l'opérateur passer en paramètre



6- Fonction " **convertir_exp_math_arbre**" : Elle insère les éléments de l'expression dans un arbre.

- **Spécification des données et des résultats :**

Le résultat est un arbre qui contient les éléments de l'expression.

- **Spécification fonctionnelle :**

Les deux premières éléments est un opérande et un opérateur, l'opérateur sera la racine et l'opérande son fils gauche.

Maintenant s'il faut insérer un opérande dans l'arbre contenant les éléments, il sera fils droit s'il est pas suivie par un opérateur.

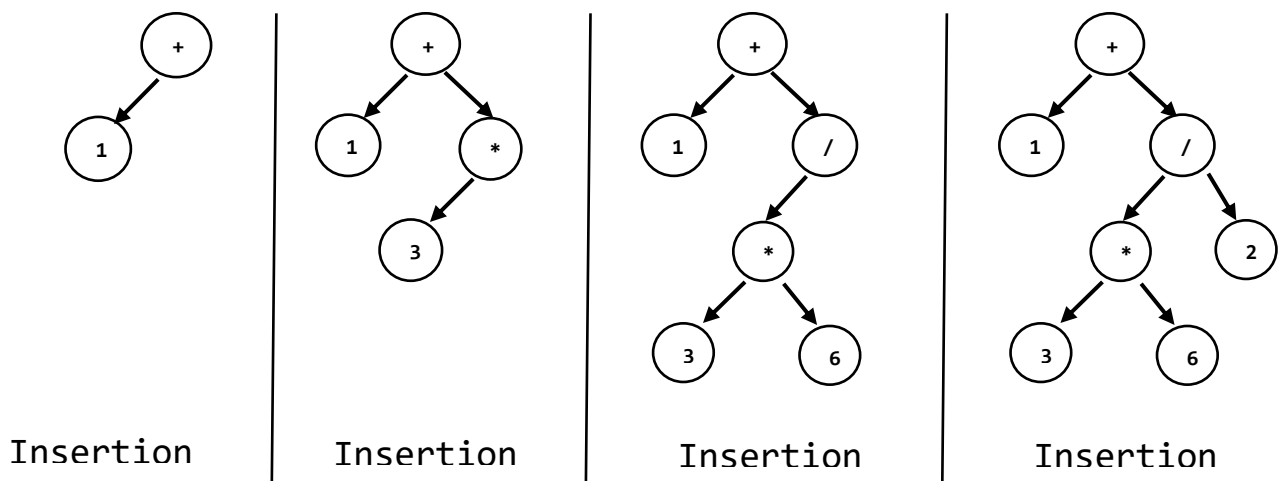
Sinon (l'opérande est suivie par un opérateur) :

L'opérateur à insérer {*,/} est le nœud courant est {+,-} alors l'opérateur devient fils droit de nœud courant et l'opérande devient le fils gauche de ce dernier.

L'opérateur à insérer {+,-} alors l'opérande devient fils gauche de nœud courant et on insère l'opérateur à la bonne place.

Tout ça pour respecter l'ordre de priorité des opérateurs

Exemple : $1+3*6/2$



7- Fonction " **Evaluation_Arb_arith**" : Elle calcule une expression arithmétique.

- **Spécification des données et des résultats :**

Un arbre.

Le résultat de calcul.

- **Spécification fonctionnelle :**

la fonction effectue les calculs récursivement sur le sous arbre gauche et droit et effectue l'opération (fils gauche racine fils droit).
Exemple : $3+2*5+4 \rightarrow 17$

III. Dossier de programmation :

~ Librairie utilisée :

```
#include<stdlib.h>
```

```
#include<stdio.h>
```

```
#include<malloc.h>
```

~ Les structures utilisées

~ **Union (U_char_float) :**

Ensemble de champs occupant tout le même emplacement en mémoire, dans ce cas :

operande: un réel (type : float) c'est la variable dans laquelle vont être stockés les opérandes.

opérateur: un caractère (type : char) c'est la variable dans laquelle vont être stockées les opérations.

```
union U_char_float
```

```
{
```

```
float operande//champ operand
```

```
char operateur;//champ operateur
```

```
};
```

~ **Structure d'un arbre :**

Nommé **Noeud** a comme champs :

Info de type Union pour stocker l'information (opérande ou opérateur).

fgche est un pointeur qui lie ce nœud avec le sous arbre gauche.

fdt est un pointeur qui lie ce nœud avec le sous arbre droit.

```
typedef struct Nd
```

```

{
    U_char_float champ_opp; //etiquette du noeud

    struct Nd* fgche;//pointeur sur le fils gauche

    struct Nd* fdt;//pointeur sur le fils droit

}Noeud; //nom de la structure

```

~ Les fonctions en C :

```

/*****Creer_Noeud*****/
/*
    Entée : U_char_float b val : la valeur à mettre dans le nouveau élément
    Sorties : le nouvel élément alloué
    La fonction prend la valeur et fait une allocation mémoire,
    et initialise cette nouvel élément avec val
*/
Noeud * Creer_Noeud(U_char_float val)
{
    Noeud *arb;

    //allocation mémoire du nouvel élément
    arb=(Noeud*)malloc(sizeof(Noeud));

    //Test si l'allocation est bien faite
    if(!arb)
    {
        printf("\nErreur d'allocation");
        exit(0);
    }
}

```



```

/*Initialisation Noeud*/

arb->champ_opp=val;

arb->fgche=NULL;

arb->fdt=NULL;

return (Noeud*) arb;

}

/*****

/*****affichage_Arb_horizontal*****/

/*

```

Entrées:

Noeud * arb: pointeur sur la racine de l'arbre.

int niveau : variable permettant de renseigner sur le
niveau de l'élément courant.

Sorties:

int : entier permet de renseigner sur l'état de la fonction

La fonction permet d'afficher l'arbre horizontalement ceci en parcourant

La partie droite de l'arbre et en l'affichant puis afficher la partie

gauche

*/

```
int afficher_Arb_horizontal(Noeud * arb,int niveau)
```

```
{
```

```
    int rst;
```

```
    //Test si l'élément courant est NULL
```

```
    if(!arb)
```

```

        return (int)0;

//afficher le sous arbre droit
rst=afficher_Arb_horizontal(arb->fdt,++niveau);

//ajouter autant de tabulation que le niveau de l'élément courant
for(int i=0;i<niveau;i++)printf("\t");

//afficher l'élément courant et retourner à la ligne
if((!arb->fgche)&&(!arb->fdt))

    printf("%f\n",arb->champ_opp.opperande);

else

    printf("%c\n",arb->champ_opp.opperateur);

//afficher le sous arbre gauche
rst=afficher_Arb_horizontal(arb->fgche,niveau);

return (int)1;

}

```

```

/*****/

```

```

/*****est_opérateur*****/

```

```

/*

```

Entée : char car : le caratere sur lequel on teste

Sorties : 1-> l'élément passer ou paramètre est un operateur

0-> sinon

```

*/

```

```

int est_opérateur(char car)

```

```
{
    return((car == '+') || (car == '-') || (car == '*') || (car == '/'));
}
```

```
/******
```

```
/******Est_numerique*****
```

```
/*
```

Entée : char car : le caractère sur lequel on teste

Sorties : la valeur numérique si le caractère est un caractère numérique

Sinon -1 ou -2 si le caractère est un opérateur de signe (+, -) sinon -3

```
*/
```

```
int Est_numerique(char cara)
```

```
{
```

```
//voir si compris entre le code 0 et 9
```

```
if(((int)'0'<=(int)cara) && ((int)cara<= (int)'9'))
```

```
    return ((int)cara - (int)'0');
```

```
if((int)cara==(int)'-') return -1;
```

```
if((int)cara==(int)'+') return -2;
```

```
    return (int)-3;
```

```
}
```

```
/******
```

```
/******priorite*****
```

```
/*
```

Entée : char op1 : l'opérateur qui se trouve dans le nœud courant

Char op2 : l'opérateur lu

Sorties :

```
*/  
  
int priorite(char op1,char op2)  
{  
    /* tester si l'opérateur qui se trouve dans le nœud courant  
    Est un + ou un -*/  
    if((op1 =='+') || (op1 == '-'))  
    {  
        // tester si l'opérateur lu est un + ou un -  
        if((op2 =='-') || (op2 == '+'))return((int)0);  
        return((int)1);  
    }  
    return((int)0);  
  
}  
  
/*****/  
  
/*****Characters_To_float*****/  
  
/*
```

Entée : char *opr : un passage par adresse d'une variable de type char

Sorties : la valeur numérique saisie

```
*/  
  
float Characters_To_float(char *opr)  
{  
    float valeur1=0,valeur2=0,rang=.1;
```

```

int unite,signe=1;

char cara;


//traitement de signe

cara=getchar();

unite=Est_numerique(cara);

switch (unite)
{
    //si le caractere est un -
    case -1: signe=-1;break;

    //si le caractere est un +
    case -2:signe=1;break;

    //si le caractère est quelque chose d'autre que le + et -
    case -3:

        printf("ERREUR 1 : expression mathématique est mal écrite");

        exit(0);

    default: valeur1=valeur1*10+unite;

}


//boucler sur la première partie du réel qui est avant le '.'
while( ( (cara=getchar()) != '.') && (cara != '\n'))
{

    unite=Est_numerique(cara);

    //le cas où le caractère n'est un caractère numérique

    if(unite < 0)break;

    valeur1=valeur1*10+unite;

```

```
}
```

```
//lecture des chiffres après la virgule
```

```
if(cara=='.')
```

```
{
```

```
    while((cara=getchar())!=(int)"\n")
```

```
    {
```

```
        unite=Est_numerique(cara);
```

```
        //le cas où le caractère n'est un caractère numérique
```

```
        if(unite < 0)break;
```

```
        valeur2+=unite*rang;
```

```
        rang/=10;
```

```
    }
```

```
}
```

```
//tester si le reel se termine par un caractère
```

```
if(unite < 0)
```

```
{
```

```
    // tester si ce caractère n'est pas un operateur
```

```
    if(!est_operateur(cara))
```

```
    {
```

```
        printf("ERREUR 2 : expression mathématique est mal écrite");
```

```
        exit(0);
```

```
    }
```

```
    *opr=cara;
```

```
}
```

```

        if(cara == '\n')*opr=cara;

        // retourner la somme des deux parties du float

        return (float)(signe*(valeur1+valeur2));

    }

/*****

/*****rendre_ancetre*****/

/*

Entée: Noeud* arbre :l'arbre ou on va se déplacer

        Noeud *element_courant[1] :le variable ou va stocker la nouvelle adresse
        de nœud courant

        Char oppérateur : l'opérateur qu'on va insérer

Sorties : l'adresse de l'arbre

*/

Noeud* rendre_ancetre(Noeud* arbre,Noeud *element_courant[1]
        ,char oppérateur)

{

    Noeud *ptr,*ptr_Svt,*NE;

    union U_char_float T;

    T.oppérateur=oppérateur;

    //création d'un nœud

    NE=Creer_Noeud(T);

```

```

//si la racine est de meme ou plus prioritaire que l'opérateur lu
if(priorite(arbre->champ_opp.opperateur,opperateur) == 0)
{
    NE->fgche=arbre;
    NE->fdt=NULL;
    element_courant[0]=NE;
    return((Noeud*)NE);
}

ptr=arbre;
ptr_Svt=arbre->fdt;
//boucler jusqu'a trouver le bon emplacement ou il faut insérer l'operateur
while(priorite(ptr_Svt->champ_opp.opperateur,opperateur) == 1)
{
    ptr=ptr_Svt;
    ptr_Svt=ptr_Svt->fdt;
}

//insérer l'opérateur a la bonne place
ptr->fdt=NE;
NE->fgche=ptr_Svt;
element_courant[0]=NE;
return((Noeud*)arbre);

}

/*****/

```



```

/*****convertir_exp_math_arbre*****/
/*
    Entée : char op1 : l'opérateur qui se trouve dans le nœud courant
           Char op2 : l'opérateur lu
    Sorties :
*/
Noeud* convertir_exp_math_arbre()
{
    int res;

    char operateur;
    float operande;
    union U_char_float elem_union;
    Noeud * arbre=NULL,* NE,*element_courant[1];

    ///l'appel de la fonction Caracters_To_float
    operande=Caracters_To_float(&operateur);
    // stocker le reel retourner
    elem_union.operande=operande;
    NE=Creer_Noeud(elem_union);

    //Si l'expression est composée d'un seul chiffre
    if(operande=="\n")
        return (Noeud*)NE;

    //Sinon on met l'operateur dans la racine et l'opérande comme fils gauche
    arbre=NE;

```

```

elem_union.opperateur=opérateur;

NE=Creer_Noeud(elem_union);

NE->fgche=arbre;

arbre=NE;

element_courant[0]=arbre;


//Traiter les éléments qui restent
while(opérateur!='\n')
{
    //l'appel de la fonction Characters_To_float
    operande=Characters_To_float(&opérateur);

    //tester si opérateur est un /n
    if(opérateur=='\n')
    {
        //on met l'operande comme fils droit
        elem_union.operande=operande;

        NE=Creer_Noeud(elem_union);

        element_courant[0]->fdt=NE;
    }
    else
    {
        //tester la priorité entre l'opérateur element_courant et l'opérateur lu
        res=priorite(element_courant[0]->champ_opp.opperateur,opérateur);

        //si element_courant est plus prioritaire
        if(res==0)
        {

```

```

        //on met l'opperande comme fils droit
        elem_union.opperande=opperande;
        NE=Creer_Noed(elem_union);
        element_courant[0]->fdt=NE;

        ///l'appel de la fonction rendre_ancetre
        arbre=rendre_ancetre(arbre,element_courant,opérateur);
    }
else
{
    /* on met l'opérateur dans la element_courant et l'opperande
    comme son fils gauche */
    elem_union.opérateur=opérateur;
    NE=Creer_Noed(elem_union);
    element_courant[0]->fdt=NE;
    element_courant[0]=NE;

    //et l'opérande comme son fils gauche
    elem_union.opperande=opperande;
    NE=Creer_Noed(elem_union);
    element_courant[0]->fgche=NE;
}

}

}

return (Noed*)arbre;
}

/*****/

```

```

/*****Evaluation_Arb_arith*****/
/*
    Entée : char op1 : un arbre
    Sorties : la valeur calculer de l'expression arithmétique
*/
float Evaluation_Arb_arith(Noeud *arbre)
{
    float opr1,opr2;

    //tester si l'arbre est vide et retourner 0
    if(!arbre)return((float)0);

    // tester s'il est une feuille (opérande)
    if((!arbre->fdt) && (!arbre->fgche))return((float)arbre->champ_opp.opperande);

    // évaluer le sous arbre gauche
    opr1=Evaluation_Arb_arith(arbre->fgche);

    // évaluer le sous arbre droit
    opr2=Evaluation_Arb_arith(arbre->fdt);

    switch(arbre->champ_opp.opperateur)
    {
        case '+':return((float)opr1+opr2);
        case '-':return((float)opr1-opr2) ;
        case '*':return((float)opr1*opr2) ;
        case '/':if( opr2 == 0)
        {
            printf("\n impossible de diviser %d sur 0 ",opr1);
            exit(0);
        }
        return((float)opr1/opr2) ;
    }
}

```

```

    }
}

/*****/

```

Chapitre2 : Résolution d'expression arithmétique à l'aide de pile

I. Analyse :

Résoudre une expression arithmétique à l'aide d'une pile.

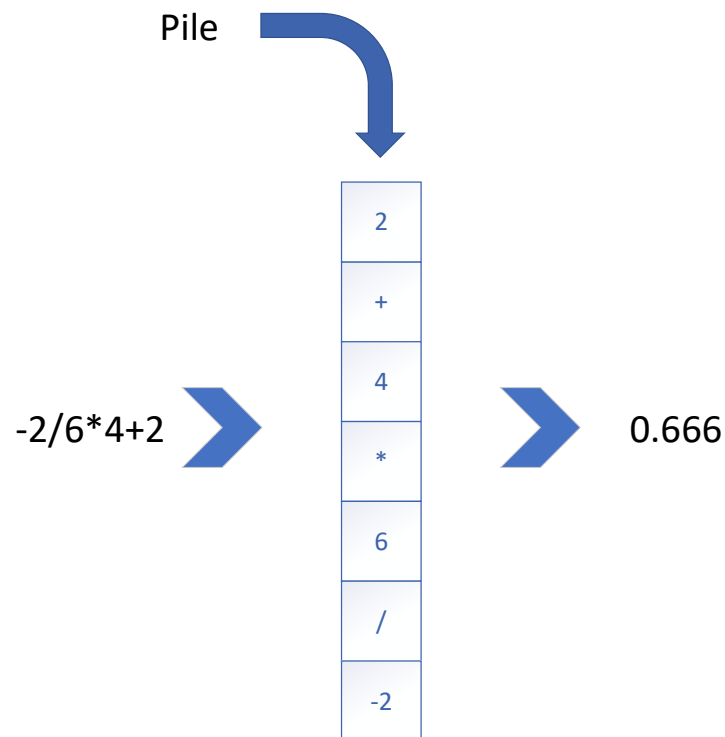


Figure 1 schématisation du problème

Idée et informations exploitables :

- _ Dans la pile, nous aurons l'ordre suivant : opérande puis opérateur.
- _ La validation d'une expression :
 - L'expression n'est pas valide s'il contient des caractères qui ne sont ni opérande ni opération.
 - Si l'expression saisie commence par une opération (*, /), ça sera non valide ;

- On ne peut pas diviser par 0 ;
- Un opérande ne peut contenir qu'une seule virgule s'il est décimal ;
- (+,-) ne peuvent pas être suivies par (* ou /) ;

II. Analyse fonctionnelle

1. Fonction 'est_opérateur' : la fonction test si le caractère est un opérateur mathématique

▪ Spécifications des données et des résultats :

On passe un caractère comme donnée, et la fonction retourne 1(le caractère est un opérateur) 0 si non.

2. Fonction 'Est_numerique' : la fonction test si le caractère est un chiffre.

▪ Spécifications des données et des résultats :

Entrée, un caractère. Si c'est un chiffre, on le retourne, sinon on retourne un entier négatif.

3. Fonction 'Characters_To_float' : Elle extrait un réel (un opérande) d'une chaîne de caractère.

▪ Spécification des données et des résultats :

Une chaîne des caractères.

Le résultat est un réel signés (un opérande), le caractère (opérateur) qui suit l'opérande {+,*/, -, \n} et un message d'erreur, le cas de '\n' c'est le cas où le réel n'est pas suivie par un opérateur.

EXEMPLE

Une chaîne des caractères numériques valide : 1 , 2- , 3/, 4* , -5.

Une chaîne des caractères numériques qui n'est pas valide : 1a, -a12 , *2 , /3 .

4. Fonction ‘‘Empiler_expression’’ : la fonction transforme une expression arithmétique en une pile d’éléments.

- Spécification des données et des résultats :
La seule donnée est le buffer qui contient l’expression arithmétique.
Le résultat est une pile dans lesquels se trouvent l’ensembles des caractères du buffer.
- Spécification fonctionnelle :
Tant que nous avons des éléments dans le buffer on les mets dans la pile. Si l’élément lu est une retour chariot, on s’arrête.

5. Fonction ‘‘eval’’ : la fonction fait l’évaluation d’une expression arithmétique composé par un seul opérateur binaire et deux opérandes

- Spécification des données et des résultats :
On passe à la fonction 2 réel , et un caractère (opérateur)
Le résultat est l’évaluation de cette opération.
Exemple :eval(2.4,’+’,0.6)-->3.0

6. Fonction ‘‘priorite’’ : La fonction donne la priorité entre opérateur, c’est-à-dire le quel devra être évalué avant l’autre.

- Spécification des données et des résultats :
On passe 2 caractères (les opérateurs).

On retourne un entier selon l’ordre de priorité des opérateurs.

Operateur 1	Operateur 2	Priorité
*	/	0
*	*	0
/	*	0
/	/	0
+	-	0
+	+	0

-	+	0
-	-	0
*	+	1
*	-	1
/	+	1
/	-	1
+	*	0
+	/	0
-	*	0
-	/	0

Priorité=0 ---> l'Opérateur 1 et l'Opérateur 2 ont soit la même priorité ou que l'Opérateur 2 a une priorité plus grande que l'opérateur 1.

Priorité=1 ---> l'Opérateur 1 a une priorité plus grande que l'Opérateur2.

7. Fonction principale ‘calculer_exp_math_pile’ : elle évalue une expression arithmétique.

▪ Spécifications des données et des résultats :

L'expression arithmétique va être insérée depuis l'entrée standard et mise dans le buffer.

A la fin du traitement, nous aurons une évaluation de l'expression qui est un réel.

Dans les cas échéant des messages d'erreur seront afficher.

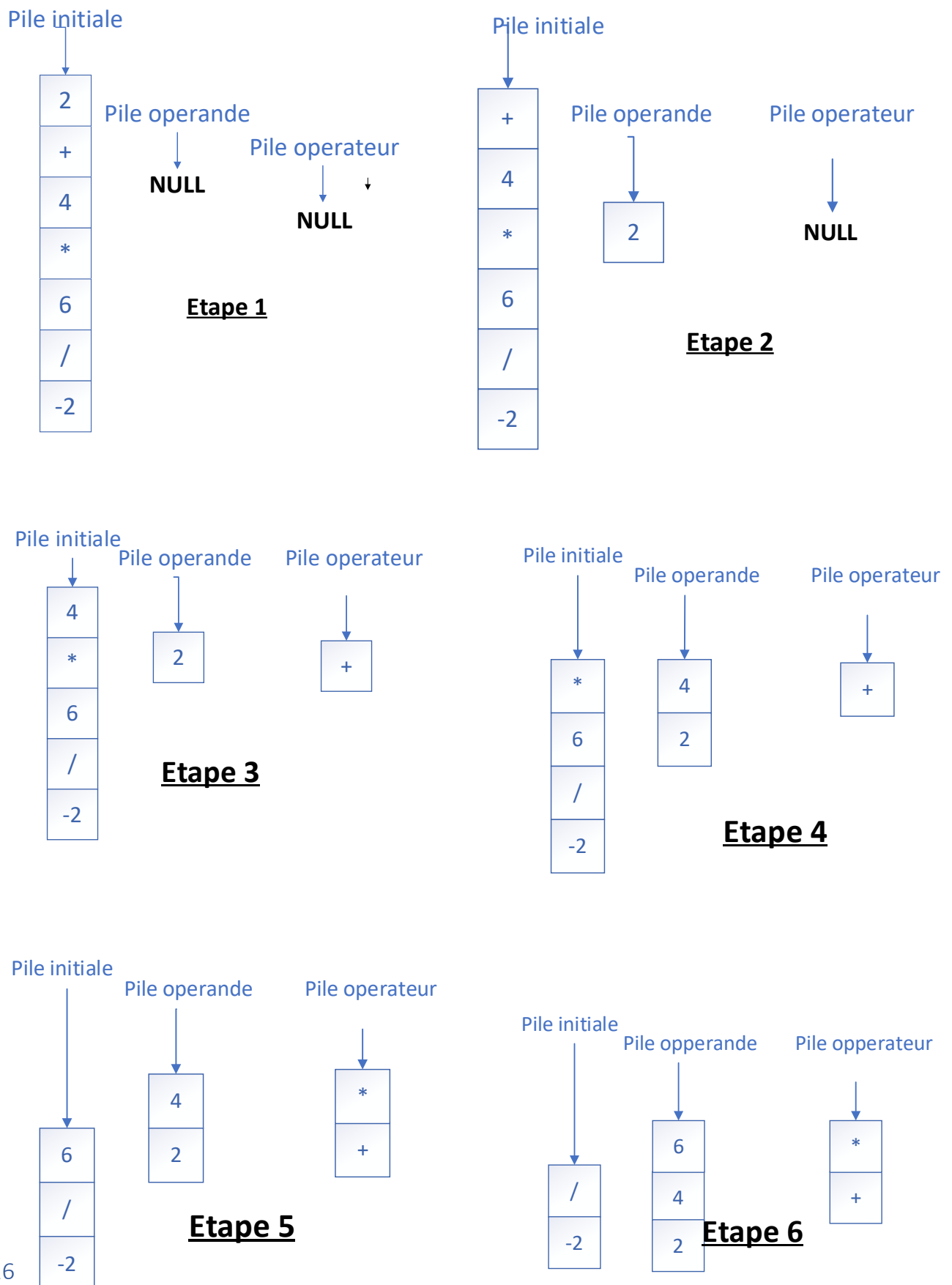
▪ Spécification fonctionnelle :

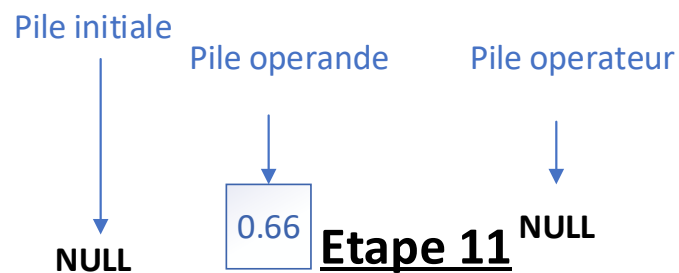
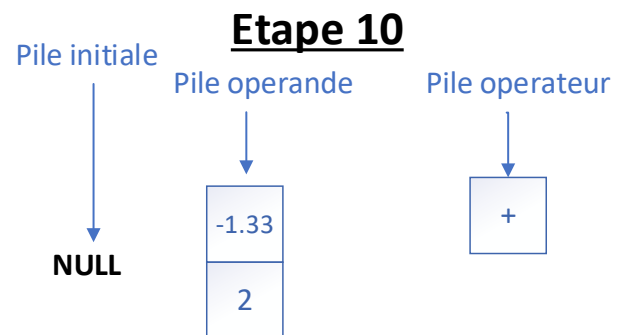
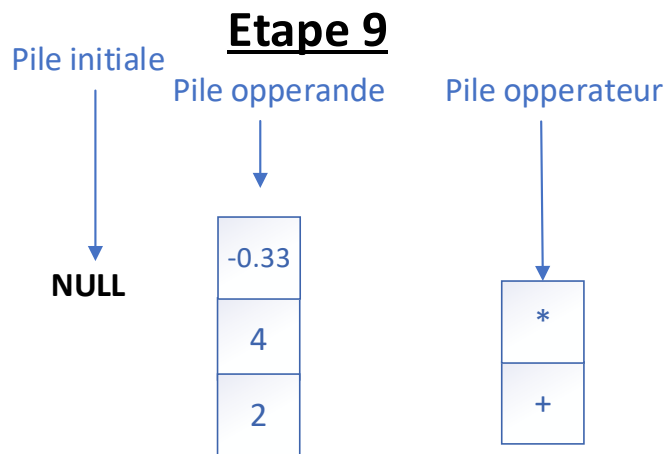
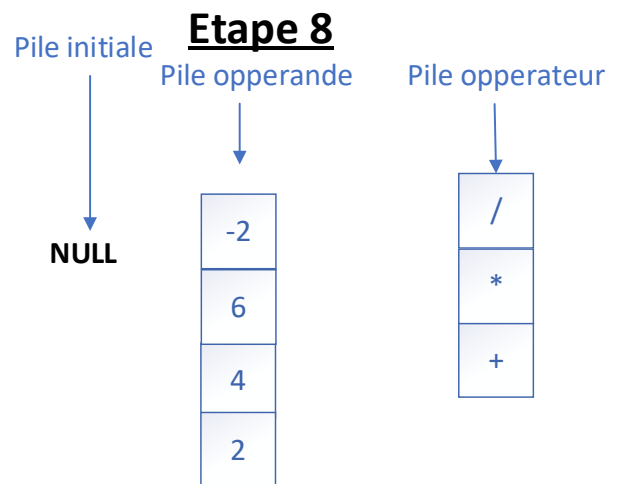
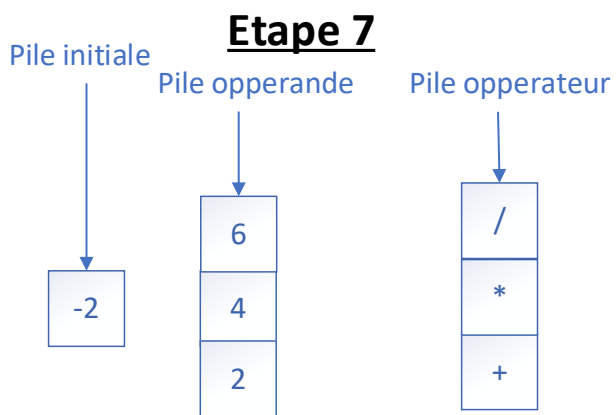
Après avoir mis l'entièreté de l'expression arithmétique dans la pile initiale, on commence à dépiler cette pile et empiler les piles opérandes et opérateurs selon la nature de l'élément.

Lorsqu'on lie un opérateur, il faut qu'on compare sa priorité avec celui qui a été empiler juste avant lui. Si l'opérateur lu a une priorité inférieure ou égale à l'opérateur en tête de la pile d'opérateur, on continue alors l'empilement dans les piles opérande et opérateur.

Si non on évalue le contenu de la pile d'opérandes et d'opérateurs, le résultat de l'évaluation sera mis dans la pile d'opérandes. Et ce n'est qu'après que nous allons empiler l'opérateur lu dans la pile d'opérateurs.

Exemple : expression arithmétique : $-2/6*4+2$





III. Dossier de programmation :

a. Les structures utilisées

Union (U_char_float) :

Ensemble de champs occupant tous le même emplacement en mémoire, dans ce cas :

Opérande : un réel (type : float) c'est la variable dans laquelle vont être stockés les opérandes.

Opérateur : un caractère (type : char) c'est la variable dans laquelle vont être stockées les opérations.

```
union U_char_float
{
    float operande//champ operand
    char operateur;//champ operateur
};
```

Structure d'une pile :

Nommé **Noeud**, elle a comme champs :

champ_opp : de type Union pour stocker l'information (opérande ou opérateur).

Svt : pointeur de type Nœud, il pointe sur le prochain nœud dans la pile.

```
typedef struct Nd
{
    U_char_float champ_opp;//champ contenant l'information
    struct Nd * svt; //Pointeur sur le prochain noeud
}Noeud;//Nom de la structure
```

Les fonctions en C :

```
/*
Nom Fonction : est_operateur
Entree : un caractere
Sortie : entier
Description : la fonction test si le caractere est
              un operateur mathematique
*/
int est_operateur(char car)
{
    return((car == '+') || (car == '-') ||
           (car == '*') || (car == '/'));
}
/*_____FIN_est_operateur_____*/
/
```

```
/*
Nom Fonction : Est_numerique
Entree : un caractere
Sortie : entier
Description : la fonction test si le caractere est
              un chiffre mathematique
*/
int Est_numerique(char cara)
{
    //voir si compris entre le code 0 et 9
    if(((int)'0' <= (int)cara) && ((int)cara <= (int)'9'))
        return ((int)cara - (int)'0');
    if((int)cara == (int)'-') return -1;
    if((int)cara == (int)'+') return -2;
}
```

```

        return (int)-3;
    }
    /*_____FIN_Est_numerique_____*/

    /*
    Nom Fonction : Characters_To_float
    Entree : un caractere un passage par adresse
    Sortie : reel
    Description : la fonction transforme le contenu de buffer en un reel
                  elle retourne aussi le dernier elemnt lu est qui n'est pas un
                  chiffre
                  ceci par le passage par adresse de la variable 'opr'
    */

    float Characters_To_float(char *opr)
    {
        float valeur1=0,valeur2=0,rang=.1;
        int unite,signe=1;
        char cara;

        //traitement de signe
        cara=getchar();
        unite=Est_numerique(cara);
        switch (unite)
        {
            //si le caractere est un -
            case -1: signe=-1;break;

```

```

        //si le caractere est un +
        case -2:signe=1;break;
        //si le caractere est quelque chose d'autre que le +
        et -
        case -3:
            printf("ERREUR 1 : expression mathematique est
                    mal ecrite");
            exit(0);
        default: valeur1=valeur1*10+unite;
    }

    //boucler sur la premiere partie du reel qui est avant le '.'
    while( ( (cara=getchar()) != '.') && (cara != '\n'))
    {
        unite=Est_numerique(cara);
        //le cas ou le caractere n'est un caractere numerique
        if(unite < 0)break;

        valeur1=valeur1*10+unite;
    }

    //lecture des chiffres apres la virgule
    if(cara=='.')
    {
        while((cara=getchar())!=(int)'\n')
        {
            unite=Est_numerique(cara);
            /*le cas ou le caractere n'est un caractere
            numerique*/
            if(unite < 0)break;

```

```

        valeur2+=unite*rang;
        rang/=10;
    }
}

if(unite < 0)
{
    if(!est_operateur(cara))
    {
        printf("ERREUR 2 : expression mathematique est mal
                ecrite");

        exit(0);
    }
    *opr=cara;
}

if(cara == '\n')*opr=cara;

// retourner la somme des deux parties du float
return (float)(signe*(valeur1+valeur2));
}

/*_____FIN_Caracters_To_float_____*/

/*
Nom Fonction : Creer_element_pile
Entree : un element de type U_char_float (union) qui est soit un
opérateur soit un réel
Sortie : pointeur de type Noeud

```


Description : la fonction creer une instance de type Noeud et la remplit avec l'union passe dans les parametres

*/

Noeud* Creer_element_pile(U_char_float element)

{

 Noeud * NE=(Noeud*)malloc(sizeof(Noeud));

 if(!NE) exit(0);

 NE->champ_opp =element;

 NE->svt=NULL;

 return (Noeud*)NE;

}

/*_____FIN_Creer_element_pile_____*/

/*

Nom Fonction : Empiler_element_pile

Entree :

 -pointeur de type Noeud

 -un element de type U_char_float (union) qui est soit un
 opérateur soit un reel

Sortie : pointeur de type Noeud

Description : la fonction empile un element(soit opérateur soit
reel) dans la pile

*/

Noeud *Empiler_element_pile(Noeud * pile,U_char_float element)

{

 Noeud *NE=Creer_element_pile(element);

 //Si la pile est vide

```

        if(!pile) return (Noeud*)NE;

        NE->svt=pile;
        return (Noeud*)NE;
    }

    /* _____FIN_Empiler_element_pile_____ */

    /*
    Nom Fonction : Depiler_element_pile
    Entree :

                -pointeur de type Noeud
    Sortie : pointeur de type Noeud
    Description : la fonction depile la tete de la pile
    */
    Noeud *Depiler_element_pile(Noeud *pile)
    {
        Noeud *ptr;
        ptr=pile;
        pile=pile->svt;
        free(ptr);
        return (Noeud*)pile;
    }

    /* _____FIN_Depiler_element_pile_____ */

```

```

    /*
    Nom Fonction : Empiler_expression
    Entree : VOID

```

Sortie : pointeur de type Noeud

Description : la fonction va prendre le contenu du buffer et le transformer en une pile d'elements

```
*/  
Noeud *Empiler_expression()  
{  
    Noeud * pile=NULL;  
    float reel;  
    char opérateur;  
    U_char_float instance1;  
  
    do{  
        //Lecture de l'operande et de l'opérateur  
        reel=Caracters_To_float(&opérateur);  
        instance1.operande=reel;  
        //Empiler l'operande dans la pile  
        pile=Empiler_element_pile(pile,instance1);  
        instance1.opérateur=opérateur;  
        /*Empiler l'opérateur a condition qu'il est pas un  
        routeur chariot*/  
        if(opérateur!='\n')  
            pile=Empiler_element_pile(pile,instance1);  
  
        //Refaire tant que l'opérateur lu n'est pas un '\n'  
    }while(opérateur!='\n');  
  
    return((Noeud*)pile);  
}  
/*_____FIN_Empiler_expression_____*/
```

```

/*
Nom Fonction :eval
Entree : trois elements de type U_char_float
Sortie : reel
Description : la fonction prends deux operandes et un operateur et
rends un reel qui est l'evaluation de cette operation
*/
float eval(U_char_float oprd1 , U_char_float opr , U_char_float
oprd2)
{
    switch(opr.opperateur)
    {
        case '+': return(oprd1.opperande + oprd2.opperande);
        case '-': return(oprd1.opperande - oprd2.opperande);
        case '*': return(oprd1.opperande * oprd2.opperande);
        case '/': if(oprd2.opperande==0)
            {
                printf("\n\nOn ne peut pas diviser par0\n\n");
                exit(0);
            }
            return(oprd1.opperande/oprd2.opperande);
    }
}
/*_____FIN_eval_____*/

```

```

/*
Nom Fonction :priorite
Entree : deux elements de type caracteres
Sortie : entier

```

Description : donne la priorite entre les opperandes

```
*/
/* opr1 -> dernier operateur
   opr2 -> operateur lu */
int priorite(char op1,char op2)
{
    if((op1 == '*') || (op1 == '/'))
    {
        if((op2 == '*') || (op2 == '/'))return((int)0);
        return((int)1);
    }
    return((int)0);
}
/*_____FIN_priorite_____*/
```

/*

Nom Fonction : calculer_exp_math_pile

Entree : VOID

Sortie : reel

Description : la focntion evalue une expression mathematique saisi sur le clavier et la stocke dans des piles pour apres l'evaluer

*/

```
float calculer_exp_math_pile()
{
    Noeud *Pile_init=NULL,*Pile_opr=NULL,*Pile_eval=NULL;
    U_char_float operande;
    float val;
```

```

//l'appel de la fonction Empiler_expression
printf("entrer une expression : ");
Pile_init=Empiler_expression();

//le cas ou on a qu'un seul operande
if(Pile_init->svt==NULL)
    return (float)(Pile_init->champ_opp.operande);

//empiler le 1er operande dans la pile d'evaluation
Pile_eval=Empiler_element_pile(Pile_eval,Pile_init->champ_opp);
Pile_init=Depiler_element_pile(Pile_init);

while(Pile_init)
{

    //Si la pile d'opérateur est vide , on continue à lire
    if(!Pile_opr)
    {
        //l'empilement de l'opérateur
        Pile_opr=Empiler_element_pile(Pile_opr,
            Pile_init->champ_opp);
        Pile_init=Depiler_element_pile(Pile_init);
        //l'empilement de l'opérande
        Pile_eval=Empiler_element_pile(Pile_eval,
            Pile_init->champ_opp);
        Pile_init=Depiler_element_pile(Pile_init);
        continue;
    }
}

```

```

/*Si les deux oppérateurs ont la même priorité , on
continu à lire*/
if(priorite(Pile_opr->champ_opp.opérateur,
            Pile_init->champ_opp.opérateur) == 0)
{
    //l'empilement de l'opérateur
    Pile_opr=Empiler_element_pile(Pile_opr,
                                  Pile_init->champ_opp);
    Pile_init=Depiler_element_pile(Pile_init);
    //l'empilement de l'opérande
    Pile_eval=Empiler_element_pile(Pile_eval,
                                   Pile_init->champ_opp);
    Pile_init=Depiler_element_pile(Pile_init);

}
//Si non on évalue le contenu de la pile d'évaluation
else
{
    operande=Pile_eval->champ_opp;
    //depiler l'opérande à la tête de pile
    Pile_eval=Depiler_element_pile(Pile_eval);
    //l'appel de la fonction eval
    val=eval(operande,Pile_opr->champ_opp,
             Pile_eval->champ_opp);
    //depiler le 2ème opérande
    Pile_eval=Depiler_element_pile(Pile_eval);
    //depiler l'opérateur
    Pile_opr=Depiler_element_pile(Pile_opr);

    operande.operande=val;
}

```

```

        //empiler le resultat dans la pile d'evaluation
        Pile_eval=Empiler_element_pile(Pile_eval,operande);

    }
}

// le cas ou on a que des operateurs de meme priorite
while(Pile_eval->svt)
{
    operande=Pile_eval->champ_opp;
    //depiler le 1er operande
    Pile_eval=Depiler_element_pile(Pile_eval);
    //l'appel de la fonction eval
    val=eval(operande,Pile_opr->champ_opp,
            Pile_eval->champ_opp);
    //depiler le 2eme operande
    Pile_eval=Depiler_element_pile(Pile_eval);
    //depiler l'operateur
    Pile_opr=Depiler_element_pile(Pile_opr);

    operande.opperande=val;
    //empiler le resultat dans la pile d'evaluation
    Pile_eval=Empiler_element_pile(Pile_eval,operande);
}

return((float)Pile_eval->champ_opp.opperande);
}

/*_____FIN_calculer_exp_math_pile_____*/

```