**Compte Rendu**



**Résolution d’expression arithmétique**

2022/2023



**Filière Ingénieur :**

**Ingénierie Logicielle et Intégration des Systèmes Informatiques**

Réalisé par :

Prénom NOM

Prénom NOM

Encadré par :

Prof. Abdelkrim BEKKHOUCHA

Table de matières

[Chapitre1 : Résolution d’expression arithmétique à l’aide d’un arbre 3](#_Toc123410124)

[I. Analyse : 3](#_Toc123410125)

[II. Analyse fonctionnelle 4](#_Toc123410126)

[III. Dossier de programmation : 7](#_Toc123410127)

[Chapitre2 : Résolution d’expression arithmétique à l’aide de pile 21](#_Toc123410128)

[I. Analyse : 21](#_Toc123410129)

[II. Analyse fonctionnelle 23](#_Toc123410130)

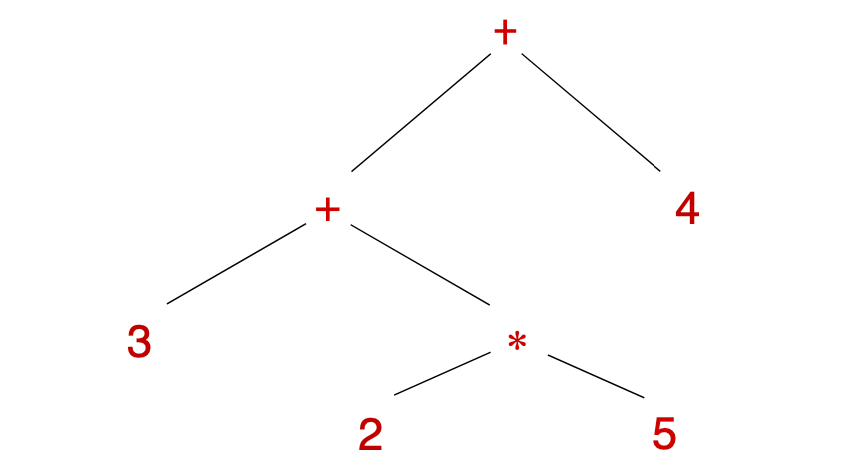
[III. Dossier de programmation : 28](#_Toc123410131)

# Chapitre1 : Résolution d’expression arithmétique à l’aide d’un arbre

## 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 saisie une chaine de caractère.
* La lecture de l’expression se fait caractère par caractère.
* Dans une expression saisie, il faut distingue 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 /) ;

## Analyse fonctionnelle

1. **Fonction ‘‘est\_operateur’’ :** 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.

1. **Fonction ‘‘Est\_numerique’’ :** la fonction test si le caractere 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 negatif.

1. **Fonction ’’** **Caracters\_To\_float’’ :** Elle extrait un réel (un opérande) d’une chaine de caractère.

* **Spécification des données et des résultats :**
* Une chaine des caractères.
* Le résultat est un réel signés (un opérande), le caractère (operateur) 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

les chaines des caractères numériques valide :

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

les chaine des caractères numériques qui ne sont pas valide :

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

1. **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’operateur 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) |

1. **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.

Apres 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

**+**

**3**

**\***

**6**

**1**

**+**

**3**

**\***

**6**

**1**

**/**

**/**

1. **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 sil est pas suivie par un operateur .

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

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

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

Tout ça pour respecter l’ordre de priorité des operateur

Exemple :1+3\*6/2

**+**

**1**

**+**

**1**

**\***

**3**

**+**

**1**

**\***

**3**

**6**

**/**

**+**

**1**

**\***

**3**

**6**

**/**

**2**

Insertion de + et 1

Insertion de 2

Insertion de / et 6

Insertion de \* et 3

1. **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 🡺 17

## 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.

**operateur**: 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 operateur).

**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;

}

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*afficher\_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\_operateur\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

Entée : char car : le caratere sur lequel on teste

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

0-> sinon

\*/

int est\_operateur(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 comprit 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'operateur lu est un + ou un -

if((op2 =='-') || (op2 == '+'))return((int)0);

return((int)1);

}

return((int)0);

}

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Caracters\_To\_float\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

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

Sorties : la valeur numérique saisie

\*/

float Caracters\_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 opperateur : l'opérateur qu'on va insérer

Sorties : l'adresse de l'arbre

\*/

Noeud\* rendre\_ancetre(Noeud\* arbre,Noeud \*element\_courant[1]

,char opperateur)

{

Noeud \*ptr,\*ptr\_Svt,\*NE;

union U\_char\_float T;

T.opperateur=opperateur;

//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 opperateur;

float opperande;

union U\_char\_float elem\_union;

Noeud \* arbre=NULL,\* NE,\*element\_courant[1];

//l'appel de la fonction Caracters\_To\_float

opperande=Caracters\_To\_float(&opperateur);

// stocker le reel retourner

elem\_union.opperande=opperande;

NE=Creer\_Noeud(elem\_union);

//Si l'expression est composée d'un seul chiffre

if(opperande=='\n')

return (Noeud\*)NE;

//Sinon on met l'opperateur dans la racine et l'opérande comme fils gauche

arbre=NE;

elem\_union.opperateur=opperateur;

NE=Creer\_Noeud(elem\_union);

NE->fgche=arbre;

arbre=NE;

element\_courant[0]=arbre;

//Traiter les éléments qui restent

while(opperateur!='\n')

{

//l'appel de la fonction Caracters\_To\_float

opperande=Caracters\_To\_float(&opperateur);

//tester si operateur est un /n

if(opperateur=='\n')

{

//on met l'opperande comme fils droit

elem\_union.opperande=opperande;

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,opperateur);

//si element\_courant est plus prioritaire

if(res==0)

{

//on met l'opperande comme fils droit

elem\_union.opperande=opperande;

NE=Creer\_Noeud(elem\_union);

element\_courant[0]->fdt=NE;

//l'appel de la fonction rendre\_ancetre

arbre=rendre\_ancetre(arbre,element\_courant,opperateur);

}

else

{

/\* on met l'opperateur dans la element\_courant et l'opperande

comme son fils gauche \*/

elem\_union.opperateur=opperateur;

NE=Creer\_Noeud(elem\_union);

element\_courant[0]->fdt=NE;

element\_courant[0]=NE;

//et l'opérande comme son fils gauche

elem\_union.opperande=opperande;

NE=Creer\_Noeud(elem\_union);

element\_courant[0]->fgche=NE;

}

}

}

return (Noeud\*)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

## 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 operateur.

## Analyse fonctionnelle

1. Fonction ‘‘est\_operateur’’ : 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.

1. Fonction ‘‘Est\_numerique’’ : la fonction test si le caractere 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 negatif.

1. Fonction ’’ Caracters\_To\_float’’ : Elle extrait un réel (un opérande) d’une chaine de caractère.

* Spécification des données et des résultats :

Une chaine des caractères.

Le résultat est un réel signés (un opérande), le caractère (operateur) 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 chaine des caractères numériques valide :1 , 2- , 3/, 4\* , -5.

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

1. 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.

1. 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

1. 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’Operateur 1 et l’Operateur 2 ont sois la même priorité ou que l’Operateur 2 a une priorité plus grande que l’opérateur 1.

Priorité=1 ---> l’Operateur 1 a une priorité plus grande que l’Opérateur2.

1. Fonction principale ‘‘calculer\_exp\_math\_pile’’ : elle évalue une expression arithmétique.

* Spécifications des données et des résultats :

L’expression arithmétique va être inséré 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 :

Apres 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 était empiler juste avant lui.

Si l’opérateur lu a une priorité inferieur ou égale à l’opérateur en tête de la pile d’opérateur, on continue alors l’empilement dans les piles opérande et operateur.

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













## Dossier de programmation :

1. **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.

**Operateur** : 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 operateur).

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 opperateur 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 focntion test si le caractere est**

**un chiffre mathematique**

**\*/**

**int Est\_numerique(char cara)**

**{**

**//voir si comprit 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 : Caracters\_To\_float**

**Entree : un caractere un passage par addresse**

**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 Caracters\_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 opperateur**

**soit un reel**

**Sortie : pointeur de type Noeud**

**Description : la fonction creer une instance de type Noeud et la remplie**

**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 opperateur**

**soit un reel**

**Sortie : pointeur de type Noeud**

**Description : la fonction empile un element(soit opperateur sois 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 opperateur;**

**U\_char\_float instance1;**

**do{**

**//Lecture de l'operande et de l'opperateur**

**reel=Caracters\_To\_float(&opperateur);**

**instance1.opperande=reel;**

**//Empiler l'operande dans la pile**

**pile=Empiler\_element\_pile(pile,instance1);**

**instance1.opperateur=opperateur;**

**/\*Empiler l'opperateur a condition qu'il est pas un routeur chariot\*/**

**if(opperateur!='\n')**

**pile=Empiler\_element\_pile(pile,instance1);**

**//Refaire tant que l'opperateur lu n'est pas un '\n'**

**}while(opperateur!='\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 opperandes et un opperateur et rends**

**un reel qui est l'evaluation de cette opperation**

**\*/**

**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 divider 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.opperande);**

**//empiler le 1er operande dans la pile d'evaulation**

**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'operateur est vide , on continu a lire**

**if(!Pile\_opr)**

**{**

**//l'empilement de l'operateur**

**Pile\_opr=Empiler\_element\_pile(Pile\_opr,**

**Pile\_init->champ\_opp);**

**Pile\_init=Depiler\_element\_pile(Pile\_init);**

**//l'empilement de l'operande**

**Pile\_eval=Empiler\_element\_pile(Pile\_eval,**

**Pile\_init->champ\_opp);**

**Pile\_init=Depiler\_element\_pile(Pile\_init);**

**continue;**

**}**

**/\*Si les deux opperateurs ont la meme priorite , on continu a lire\*/**

**if(priorite(Pile\_opr->champ\_opp.opperateur,**

**Pile\_init->champ\_opp.opperateur) == 0)**

**{**

**//l'empilement de l'operateur**

**Pile\_opr=Empiler\_element\_pile(Pile\_opr,**

**Pile\_init->champ\_opp);**

**Pile\_init=Depiler\_element\_pile(Pile\_init);**

**//l'empilement de l'operande**

**Pile\_eval=Empiler\_element\_pile(Pile\_eval,**

**Pile\_init->champ\_opp);**

**Pile\_init=Depiler\_element\_pile(Pile\_init);**

**}**

**//Si non on evalu le contenu de la pile d'evaluation**

**else**

**{**

**operande=Pile\_eval->champ\_opp;**

**//depiler l'operande a la tete 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 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);**

**}**

**}**

**// 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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/**