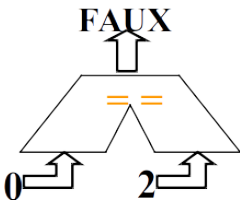
Algorithmique & Programmation en C

CALCULER

Ressources :

Diapos CALCULER



Objectif :

- Quels calculs peuvent être réalisés dans un programme ?
- Comment écrire une instruction de calcul ?
- Comment le CPU exécute-t-il une instruction de calcul ?

1. DEFINITION

A partir de données en Entrée, un programme effectue un traitement et produit des données en Sortie (affichage des résultats). Ce traitement est souvent un **calcul** sur les données d'Entrée (déclarées et initialisées par affectation ou saisie clavier).

Un calcul s'écrit à partir d'une **formule de calcul**, qui peut-être arithmétique, binaire ou logique :

- ✗ une formule arithmétique utilise des opérateurs arithmétiques pour effectuer des calculs en base décimale (addition, soustraction, multiplication, division, modulo) ;
- ✗ une formule binaire utilise des opérateurs binaires pour effectuer des calculs en base 2 (inverseur, ET binaire, OU binaire, OU exclusif) ;
- ✗ une formule logique utilise des opérateurs logiques (de comparaison -supérieur, inférieur, égal, différent- et booléens -NON logique, ET logique, OU logique-).

Les équations logiques, utiles pour les conditions logiques de certaines instructions, sont vues dans la partie traitant des instructions alternatives. Les **formules arithmétiques permettent d'écrire les calculs courants en base 10** (travaillant sur d**es entiers ou** des **réels**) ; elles peuvent-être composées :

- ✗ de **fonctions mathématiques standards** : fonctions trigonométriques -ex. fonction cosinus cos(theta)- ou arithmétiques -ex. fonction racine carrée sqrt(a)- de la bibliothèque standard *math.h*.
- ✗ d'**expressions arithmétiques** construites avec des **opérateurs arithmétiques** :

En ALGO	Nom	En C
+	Addition	+
-	Soustraction	-
x	Multiplication	*
/	Division	/
%	modulo	%

Exemple d'expressions arithmétiques :

- ✗ (axb+3.) / (c-5.) : **les points, après les valeurs, indiquent que les valeurs sont de type réel**.
- ✗ a%b : le modulo donne le reste de la division euclidienne de a par b (**a et b sont des entiers !**).

2. ALGORITHME

Une instruction de calcul est une instruction qui permet de demander au CPU de faire effectuer un calcul par l'UAL (Unité Arithmétique et Logique). Elle s'écrit à partir d'une affectation, le résultat du calcul devra être affecté à une variable résultat dans la RAM. La **variable résultat doit toujours être avant le signe d'affectation** = ; **une formule de calcul se trouve à droite du signe d'affectation** :

```
/*****************************************************************************
                        INSTRUCTION DE CALCUL
*****************************************************************************/
ALGO  Calculer
        VAR     a=5.3, b= 2., r=18.5 :  réels double précision  // opérandes de calcul
                moy, aire :  réels double précision  // moyenne et aire du disque
        CONST   PI= 3.14 :  réels double précision
DEBUT
        // nomVariableResultat= Formule de Calcul
        moy= (a+b)/2.                   // moyenne de a et b
        aire= PI x r²       // aire du disque de rayon r
FIN
```

3. CODAGE EN LANGAGE C

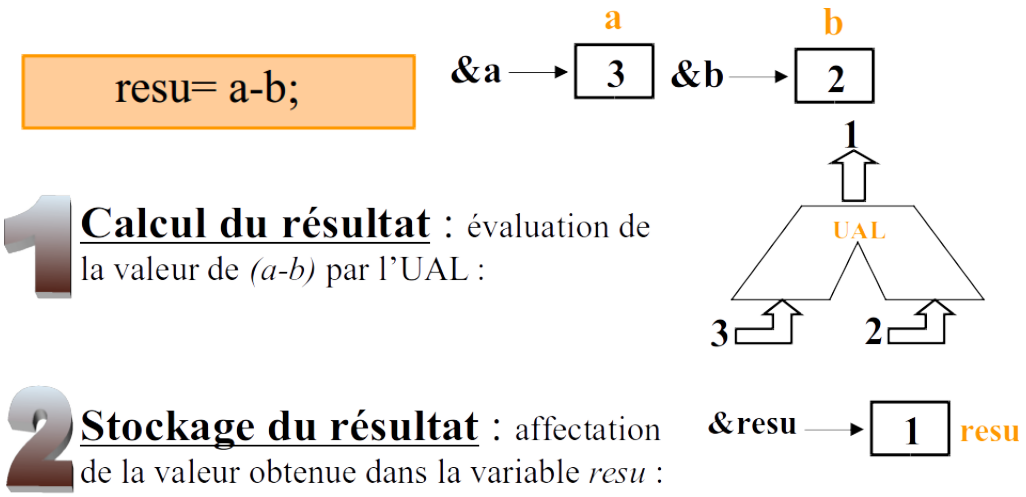
La traduction en C est immédiate, il faut adapter les opérateurs et utiliser les fonctions standards avec la bibliothèque mathématique :

```
/*****************************************************************************
                        INSTRUCTION DE CALCUL
*****************************************************************************/
#include <math.h>

int main()
{       double a=5.3, b= 2., r=18.5;  // opérandes de calcul
        double moy, aire;  // moyenne et aire du disque
        const   double PI= 3.14;

        // nomVariableResultat= Formule de Calcul
        moy= (a+b)/2.;          // moyenne de a et b
        aire= PI * pow(r,2.);     // aire du disque de rayon r
}
```

Le CPU, exécutant une instruction de calcul, réalise deux étapes :

1. iI fait d'abord **effectuer le calcul par l'UAL** (**formule de calcul à droite du signe d'affectation**) ; les valeurs utiles pour le calcul sont fournies à l'UAL, qui évalue alors le résultat du calcul.
2. Il **attribue**, ensuite, **la valeur du résultat fourni par l'UAL, à l'espace mémoire de la variable placée à gauche du =** (valeur résultat affectée à la variable résultat)**.** Pour rappel : l'exécution de l'affectation entraîne une écriture dans la RAM (qui provoque l'écrasement de la valeur qui était préalablement stockée dans l'espace RAM).

L'exemple de la décomposition de l'exécution de l'instruction de soustraction a-b est donné ici :



Concernant la première partie, pendant laquelle l'UAL évalue le calcul, les règles d'évaluation classique suivantes sont appliquées successivement :

1. L'ordre de calcul dépend des parenthèses (elles forcent les priorités). Ex. : (4+2)x2 vaut 12.
2. L'ordre de calcul dépend, ensuite, de la priorité des opérateurs. Ex. : 4+2x2 vaut 8. Le tableau suivant précise l'ordre de priorité de l'ensemble des opérateurs du langage C :

Priorité	Opérateur
1	()
2	NON (*non logique*) — (*complément à 1*)
3	x (*multiplication*) / (*division*) % (*modulo*)
4	+ (*addition*) - (*soustraction*)
5	< > ≤ ≥
6	== (***égalité***) ≠ (*différent*)
7	. (*et binaire*)
8	⊕ (*ou exclusif*)
9	+ (*ou binaire*)
10	ET (*et logique*)
11	OU (*ou logique*)
12	= (***affectation***)

ATTENTION à ne pas confondre les opérateurs = (affectation) et == (égalité) !!

- = permet d'écrire dans la RAM (une valeur est attribuée à une donnée)
- == permet de lire dans la RAM (deux valeurs sont comparées)

3. Pour un même niveau de priorité, l'évaluation de l'expression se fait de gauche à droite (à partir de l'opérateur d'affectation =). Ex. : 4/2/2 vaut 1.

4. ATTENTION TYPAGE DES DONNEES & CALCULS

Des erreurs de calcul peuvent se produire avec la division si le programmeur ne fait pas attention aux types de données utilisés. Le programmeur écrit a/b dans son code que ce soit pour faire une division entière (division euclidienne avec reste et dividende) ou une division réelle (le résultat a une partie décimale). Il doit donc savoir ce qui indique au CPU et à l'UAL la division à réaliser.

a- Définition division entière ou réelle

Donc, en écrivant a/b, le CPU va-t-il réaliser une division entière ou réelle ? La règle est :

- ✗ **Si a ET b sont des entiers, l'UAL effectue la division entière de a par b**. Le résultat du calcul est le quotient, entier résultat de l'application de l'opérateur / (*division*), et le reste, entier résultat de l'application de l'opérateur % (*modulo*).

```
/*************************************************************************
                        DIVISION ENTIERE
*************************************************************************/
int main()
{       int     a=3, b=2;   // opérandes entiers
        int     div, reste;  // quotient et reste

        // division entière
        div= a / b;
        reste= a % b;
}
```

a (3) | b (2)

reste (1) | div (1)

- ✗ **Si a OU b est un réel, l'UAL effectue la division réelle de a par b**. Le résultat du calcul est un nombre décimal, réel résultat de l'application de l'opérateur / (*division*).

```
/*************************************************************************
                        DIVISION REELLE
*************************************************************************/
int main()
{       float   a=3., b=2.;     // opérandes réelles
        float   div;            // résultat de la division réelle

        // division
        div= a / b;
}
```

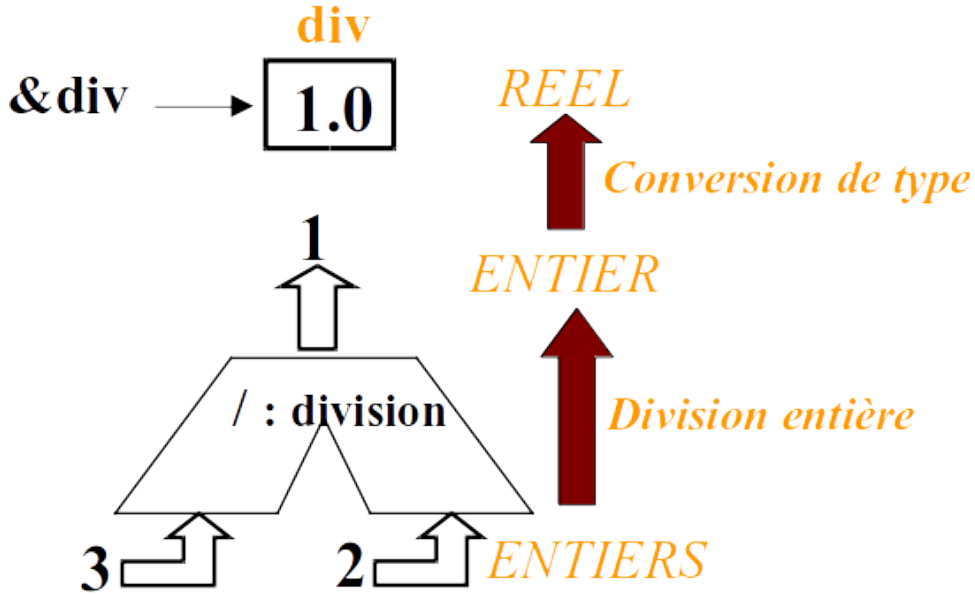
div (1.5) = a (3) / b (2)

b- Erreurs de compatibilité de types

Les exemples suivants illustrent des erreurs de compatibilité de types lorsque le programmeur mélange les types de données dans les calculs :

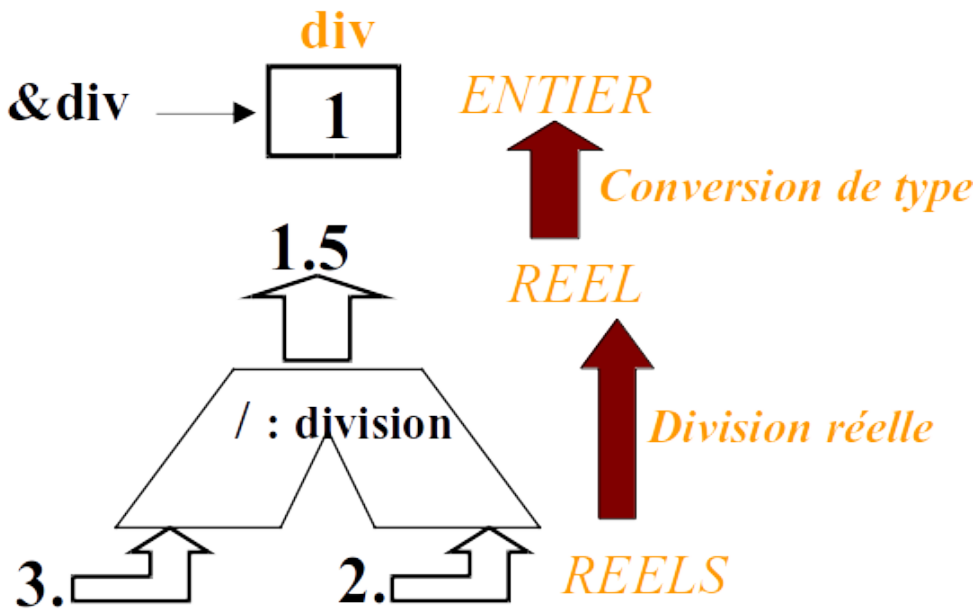
✗ Les opérandes sont entiers et le résultat est réel : si le programmeur espérait obtenir le résultat de la division réelle, c'est raté ! La division entière est réalisée car a et b sont entiers ; le résultat a une décimale nulle, puisque la valeur finale est convertie en réel :

```
/*************************************************************************
                        ERREUR TYPES 1
*************************************************************************/
int main()
{       int     a=3, b=2;   // opérandes entiers
        float   div;  // résultat réel

        div= a / b;       // division
}
```



✗ Les opérandes sont réels et le résultat est entier : si le programmeur espérait obtenir le résultat de la division réelle, c'est raté ! La division réelle est bien réalisée car a et b sont réels ; par contre, le résultat est tronqué (partie décimale supprimée), puisque la valeur finale est convertie en entier :

```
/*****************************************************************************
                        ERREUR TYPES 2
*****************************************************************************/
int main()
{       float   a=3., b=2.;     // opérandes réels
        int     div;            // résultat entier

        div= a / b;     // division
}
```



Règle de programmation : ***compatibilité de types***

pour éviter des erreurs de calculs, utiliser, au maximum, des variables de même type dans un calcul arithmétique !!