Chapitre 2 : Algorithmes séquentiels simples

0. Définition : Notion d'algorithmique
1. Parties d'un algorithme
2. Les données : variables et constantes
3. Types de données
4. Opérations de base
5. Instructions de base :

- Affectations
- Instructions d'entrée sorties

5. Construction d'un algorithme simple
6. Représentation d'un algorithme par un organigramme
7. Traduction en langage C

2.1 **Définition** (**Langage algorithmique**)

Le langage algorithme est un mode d'expression des algorithmes qui nous rapproche des langages machine, sans être lui-même exécutable, sauf manuellement. Cependant, ce langage est important puisqu'il possède les qualités suivantes :

- Il est proche des langages de programmation
- Il permet d'exprimer de façon rapide les algorithmes
- Il est en même temps indépendant de tout langage de programmation réel.

Nous présentons ci-après, les éléments essentiels de ce langage.

2.2 Structure générale d'un programme dans le langage algorithmique

Cette structure est représentée dans la Fig. 1.3.

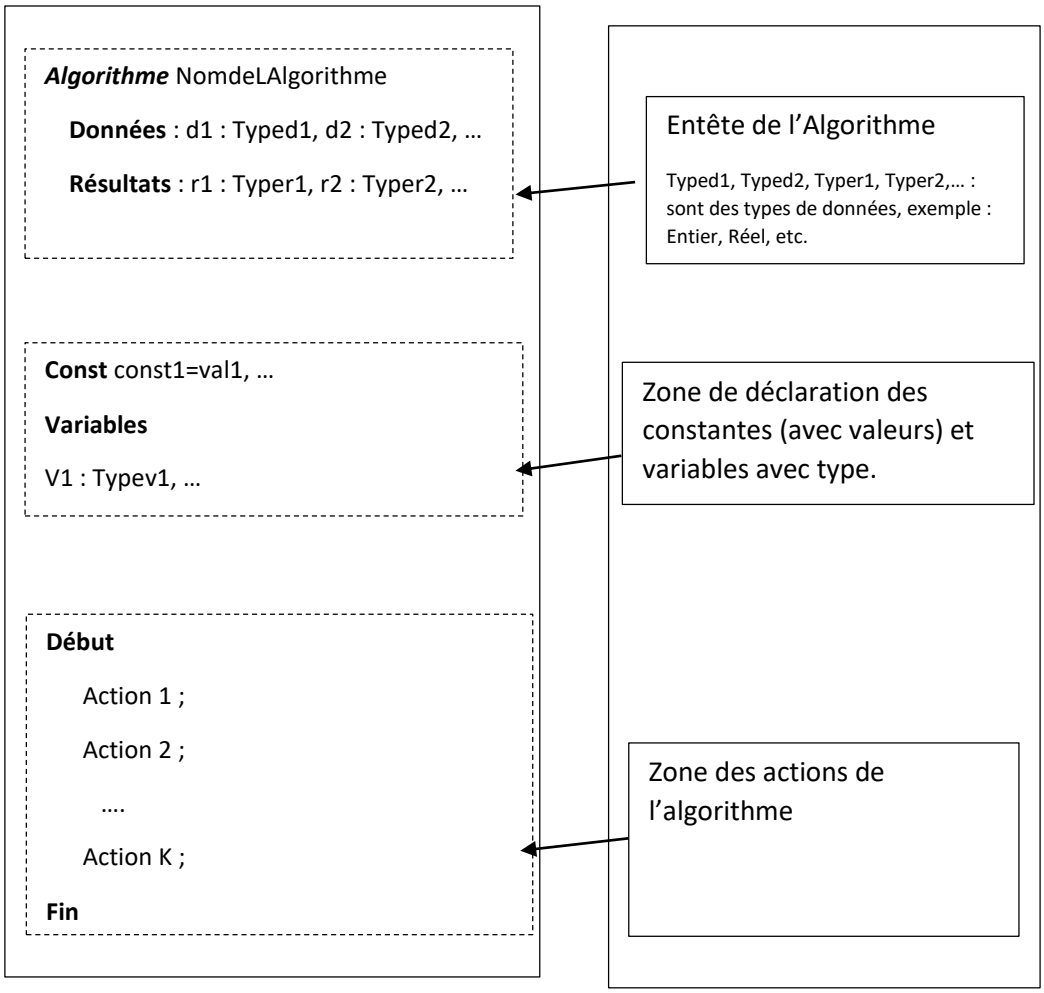
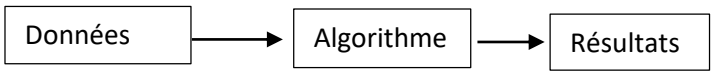


Figure 2.1 Structure générale d'un algorithme dans le langage algorithmique

Comme le montre la Figure 2.1, un algorithme est constitué de trois parties :

- L'entête de l'algorithme : *Nomdelalgorithme* + *Données* + *Résultats* : Cette partie représente la définition de l'algorithme (*Spécification* ou *ce que fait l'algorithme*) :



* Zone des déclarations (Constantes et Variables).

* La zone des actions de l'algorithme.

Ces deux zones constituent ensemble le corps de l'algorithme (ou *Comment* l'algorithme mène des données aux résultats)

2.2 Types de données de base :

Les types de données sont un concept essentiel pour tous les langages de programmation. Ils permettent de spécifier la nature des données (entrée, sortie, variables, constantes) et aussi les opérations définies sur chaque type. Nous recensons alors les types de base communs à tous les langages de programmation : (https://web.maths.unsw.edu.au/~lafaye/CCM/c/ctype.htm)

***Le type Entier* (*int en C*)** : Il représente les nombres entiers signés. Exemples : -12, 0, +14, etc. Il est représenté en mémoire selon le mode de la **virgule fixe :**

Type	**Nombre de bits**	**Intervalle de représentation**
short int	**16 (2 octets)**	**-32 768 à +32 767 (-2^{15} à $+2^{15}-1$)**
int	**32 (4 ocyets)**	**-2 147 483 648 à + 2 147 483 647 (-2^{31} à $2^{31}-1$)**
long int	**64 (8 octets)**	**-2 147 483 648 à + 2 147 483 647 (-2^{63} à $2^{63}-1$)**

Les opérations arithmétiques applicables sur ce type sont :

Opération	**Symbole**	**Arité (nombre d'opérandes)**	**Exemple : a,b : Entier (a=5,b=2)**	**Type du Résultat**	**Valeur du résultat**
Addition	**+**	Binaire	a+b	Entier	7
Soustraction	**-**	Binaire	a-b	Entier	3
Multiplication	*****	Binaire	a*b	Entier	10
Division entière	**%**	Binaire	a % b	Entier	2

Il y a aussi les opérations arithmétiques unaires : C'est-à-dire, les opérations arithmétiques qui s'appliquent sur un seul opérande

Opération	**Symbole**	**Arité (nombre d'opérandes)**	**Exemple : a,b : Entier (a=5,b=2)**	**Type du Résultat**	**Valeur du résultat**
Moins unaire	**-**	*Unaire*	-a	Entier	-5
Incrémentation **préfixée**	**++**	*Unaire*	++a	Entier	6
Incrémentation **Postfixée**	**++**	*Unaire*	a++	Entier	6
Décrémentation **préfixée**	**--**	*Unaire*	--a	Entier	4
Décrémentation **postfixée**	**--**	*Unaire*	a--	Entier	4

Précisions : Pour les opérations unaires ++ (*resp*. --), en mode préfixé : la valeur de l'opérande (exemple : a) est modifiée : a = a+1 (*resp*. a= a-1), avant d'être utilisée. En mode postfixé, il y a utilisation de l'opérande, puis modification de sa valeur : a = a+1 (*resp*. a= a-1).

Exemple illustratif des opérations unaires en C : (Exemple à exécuter dans les séances de TPs).

```
#include <stdio.h>
int main() {
 int a=5,b=2;
 int c,d,e,f;
 printf("   a= %d   b=%d",a,b);
 c=++a+b;
 printf("\n  c= ++a + b donne    a= %d  b= %d  c= %d ",a,b,c);
 d=a++ +b;
 printf("\n  d= a++ + b donne   a= %d  b= %d  d= %d ",a,b,d);
 e=--a +b;
 printf("\n  e= --a + b donne  a= %d  b= %d  e= %d ",a,b,e);
 f=a-- +b;
 printf("\n  f= a-- + b donne  a= %d  b= %d  f= %d ",a,b,f);
 return 0;
 }
```

Résultats :

```
 a= 5   b=2
 c= ++a + b donne    a= 6  b= 2  c= 8
 d= a++ + b donne    a= 7  b= 2  d= 8
 e= --a + b donne    a= 6  b= 2  e= 8
 f= a-- + b donne    a= 5  b= 2  f= 8
```

Opérations de **comparaison** sur le type entier :

Opération	Symbole	Type du résultat	Exemple (a,b : entier ; A= 5, b=2)
Supérieur à	>	Logique	a>b : vrai
Inférieur à	<	Logique	a<b : Faux
supérieur ou égal	>=	Logique	a>=b : Vrai
Inférieur ou égal	<=	Logique	a<=b : Faux
Egal	==	Logique	a==b : faux
Différent	!=	Logique	a !=b : Vrai

Ces opérations sont illustrées par l'exemple suivant en langage C : (Exemple à exécuter aux TPs).

```
#include <stdio.h>
int main() {
 int a=5,b=2;
 int c,d,e,f,g,h;
 printf("   a= %d   b=%d",a,b);
 c=(a>b);
 printf("\n  c= (a>b)   donne    a= %d  b= %d  c= %d ",a,b,c);
 d=(a<b);
 printf("\n  d= (a<b)   donne    a= %d  b= %d  d= %d ",a,b,d);
 e= (a>=b);
 printf("\n  e= (a>=b)  donne    a= %d  b= %d  e= %d ",a,b,e);
 f=(a<=b);
 printf("\n  f= (a<=b)  donne    a= %d  b= %d  f= %d ",a,b,f);
 g=(a==b);
 printf("\n  g= (a==b)  donne    a= %d  b= %d  g= %d ",a,b,f);
 h=(a!=b);
 printf("\n  h= (a!=b)  donne    a= %d  b= %d  h= %d ",a,b,h);
 return 0;
 }
```

Résultats :

```
 a= 5   b=2
 c= (a>b)   donne    a= 5  b= 2  c= 1
 d= (a<b)   donne    a= 5  b= 2  d= 0
 e= (a>=b)  donne    a= 5  b= 2  e= 1
 f= (a<=b)  donne    a= 5  b= 2  f= 0
 g= (a==b)  donne    a= 5  b= 2  g= 0
 h= (a!=b)  donne    a= 5  b= 2  h= 1
```

Le Type Réel **(float *en C*)** : Il représente les nombres réels signés. Exemples : +12.0, 0.56, -14.6, etc. Il est représenté en mémoire selon le mode **Virgule Flottante**.

Type	**Nombre de bits**	**Intervalle de représentation**
float	**32 (4 octets)**	**-3.40282e+38 à -1.17549e-38** **+1.17549e-38 à +3.40282e+38**
double	**64 (8 octets)**	**-1.79769e+308 à -2.22507e-308** **+2.22507e-308 à +1.79769e+308**
long double	**128 (16 octets)**	**$-3.4*10^{4932}$ à $-3.4*10^{-4932}$** **$+3.4*10^{-4932}$ à $+3.4*10^{4932}$**

Les calculs effectués sur ce type sont en général **approchés**. Les opérations définies sur ce type sont les opérations arithmétiques en mode virgule flottante + - * / etc., et les comparaisons **>, <**.

Le Type Caractère **(*Char en langage C*)** : Il représente les symboles : Alphabet : 'a'..'z' 'A'..'Z'), Chiffre : '0'..'9', les symboles mathématiques et spéciaux : & » ( ) [ ] , ; @ etc. Ces données sont représentées en mémoire selon le code ASCII. Les opérations définies sur ce type sont les comparaisons : ==, >=, <=, !=, etc.

Type	Nombre de bits	Intervalle de représentation
char	8 (1 octet)	-128 à +127
Unsigned char	8 (1 octet)	0 à 255

Le Type Booléen ou Logique (Boolean) : Ce type est défini sur l'ensemble {*True* , *False*}. Les opérations définies sur ce type sont OR, AND, NOT.

Table 1.2 : Opérateurs logiques OU, ET, NOT

Expression Logique	Valeurs : V= Vrai (True) ; F=Faux (False)			
A	V	V	F	F
B	V	F	V	F
A OR B	V	V	V	F
A AND B	V	F	F	F
Not A	F		V	

En langage C :

Valeur logique	**En langage C (Bib : #include<stdbool.h>**	**Valeur en C**
Vrai	*True*	*1 (ou True)*
Faux	*False*	*0 (ou False)*

Exemple : (Exemple à exécuter aux TPs).

```
#include <stdio.h>
#include<stdbool.h>    // Cette bibliothèque est nécessaire pour True et False
int main()
{
int c,d;
bool x=false;
bool y=true;
c=(x==true);
printf("x=false -> c=(x==true) = %d \n",c);
d=(y==true);
printf("y=true -> d=(y==true)   =%d",d);
return 0;
}
```

Résultats :

```
x=false -> c=(x==true) = 0
y=true -> d=(y==true)   =1
```

Opérateurs logiques en langage C :

Opérateur	Symbole en C	Exemple : (a=5, b=3)	Résultat de l'expression logique
AND	&&	(a>1) && (b<5)	True
OR	||	(a>10) || (b>5)	False
Not	!	!(a>10)	True

```
#include<stdio.h>
int main(){
int c,d,e;
      int a=5, b=3;
   c=((a>1) || (b<5));
      printf("Cas 1 :  c=((a>1) || (b<5)) -> c=%d\n",c);
   d=((a>10) || (b>5));
      printf("Cas 2 : d=((a>10) || (b>5)) -> d=%d \n",d);
   e=(!(a>10) ) ;
   printf("Cas 3 : e=(!(a>10) )              -> e=%d \n",e);
   return 0;
}
```

Sortie :

```
Cas 1 :  c=((a>1) || (b<5))   -> c=1
Cas 2 : d=((a>10) || (b>5)) -> d=0
Cas 3 : e=(!(a>10) )          -> e=1
```

2.3 Expressions Arithmétiques et Logiques et notion de priorité des opérateurs :

Plusieurs variables et/ou constantes peuvent être combinées pour obtenir des expressions ***arithmétiques***, ou ***logiques***, ou ***arithmétiques et logiques***. Dans ce cas, il faut savoir que toute expression est évaluée de gauche à droite, selon des ***priorités préétablies***. Ces priorités peuvent être dépassées par l'utilisation de parenthèse :

Voici l'ordre de priorité des opérations arithmétiques et logiques : (Table 2.1)

Table 2.1 Priorité des opérateurs arithmétiques et logiques

Opération	Priorité	Expression 1	Ordre 1 : Résultat	Expression 2	Ordre 2 : Résultat
Parenthèses : ()	1	2*4+5	*, + : 13	2*(4+5)	+,* : 18
Fonctions :	2	2*4+5	*, + : 13	$2*\sqrt{4+5}$	+, √ , * : 6
Moins Unaire, NOT	3	-2+5	- unaire, +binaire : 3	NOT (-2>3)	*-unaire, >, Not True*
*, /, DIV, MOD, AND	4	(37 DIV 5 > 3) AND (37 MOD 5 > 3)	DIV, >, MOD, <, AND: False	(37 DIV 5 > 3) AND (37 MOD 5 >0)	DIV, >, MOD, <, AND: True
+, -, OR	5	((37 + 5) > 48) OR ((37 – 5) < 22)	+, >, -, <, OR: False	(5 < (48-37)) OR (37 < (22+5))	-, >, +, >, OR: True
==, !=, <, >, <=, >=	6	(37 + 5) == (47 - 5)	+, -, == : True	(37 + 5) != (47 - 5)	+, -, != : False

2.4 Règles de conversion de types (Langage C) :

Il faut noter deux façons de conversion de types :

a. **Conversion implicite** (*automatique*) : Elle est opérée dans les expressions arithmétiques mixant des variables et/ou constantes de types différents (réel, entier) ou dans une affectation. En général, l'expression est convertie automatiquement au type réel (le **type le plus fort**).

Exemple : (en C, Expr2.c) (A exécuter dans les séances de TPs).

Expression avec types	Valeur finale avec type
X=2.5*4 (Réel, entier)	10.0 (réel)
Y=(23 Div 5)*3.5 (entier, entier, réel)	14.0 (réel)
a=x ; (entier, réel) b=y ; (entier, réel)	a :10 ; b : 14 (entier, entier)
v=a ; (réel, entier) w=b ; (réel, entier)	v : 10.0 w : 14.0

```
#include<stdio.h>
int main() {
  float x,y;
  int a,b;
  float v,w;
  x=2.5*4;
  y=(23 / 5)*3.5;
  printf("Expression 1: 2.5*4 = %f \n",x);
  printf("Expression 2: (23 div 5)/2 = %f \n",y);
  a=x; b=y;
  printf("a= %d  b=%d \n", a,b);
  v=a; w=b;
  printf("v=  %f  w=%f  \n",v,w);
 return 0;
}
```

Sortie :

Expression 1: 2.5*4 = 10.000000

Expression 2: (23 div 5)/2 = 14.000000

a= 10 b=14

v= 10.000000
w=14.000000

b. **La conversion de type explicite** (par le programmeur): Cette conversion est opérée sur demande explicite du programmeur pour passer du type réel au type entier ou versa.

- Passage du type réel au type entier : soit **x de type *float***, alors **(*int* x)** est la conversion de x en type ***int***
- Passage du type entier au type réel : soit **y de type *int***, alors **(*float* y)** est la conversion de y en type ***float***

Exemple : (en C, expr3.c) (A exécuter aux séances de TPs)

```
#include<stdio.h>
main() {
  float x=4.5;
  int y=17;
  printf("Cas1:  x*y= %d \n",(int) x*y);
  printf("Cas2: x*y=  %f  \n",x*(float) y);
}
```

```
Sortie :
Cas1:  x*y= 68
Cas2: x*y=  76.500000
Appuyez sur une touche pour
continuer...
```

2.5 Les actions : Les actions représentent la partie dynamique de l'algorithme ou programme. Nous recensons alors les actions simples et les actions de contrôle.

2.5.1 Les actions simples : Ce sont les actions qui peuvent agir sur les valeurs des variables, sans modifier explicitement le point d'exécution du programme (***Compteur Ordinal : CO : Registre contenant l'action à exécuter***). Ces actions sont : La **lecture**, **l'écriture** et **l'affectation** ou l'assignation.

2.5.1.a Actions de lecture :

Syntaxe en langage algorithmique : ***Lire (v1, v2, …)***

Syntaxe en langage C : ***scanf(&v1, &v2, …)***

Effet : Cette action saisie la valeur d'une ou plusieurs variable de l'extérieur (en entrée : Clavier, Disque, etc.) et la stocke dans l'emplacement mémoire réservée à la variable correspondante. (Fig.1.4)

Remarques :

- **E**n langage C, **printf** et **scanf** nécessitent l'inclusion de la bibliothèque d'E/S : **<stdio.h>**
- Le symbole **&** est pour indiquer un **accès par adresse à la variable**. Autrement, la valeur de la variable n'est pas modifiée.

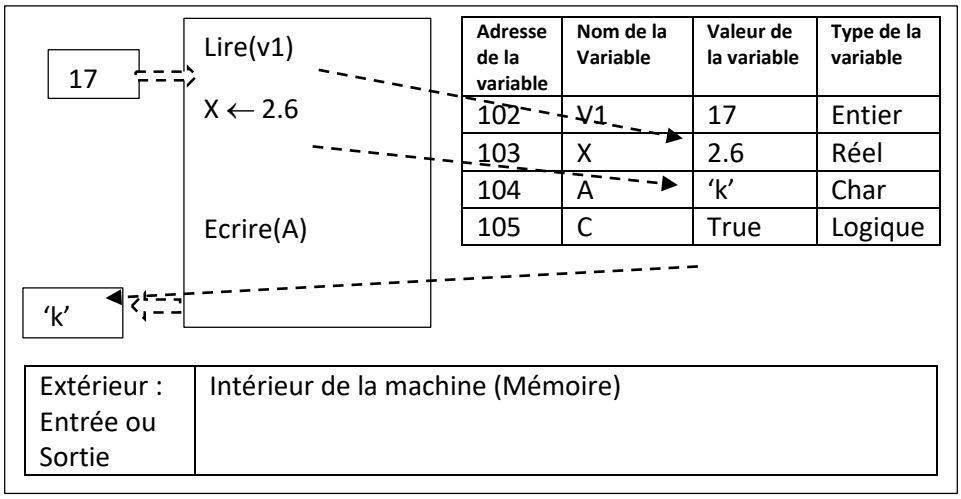


Figure 1.4 Actions simples : Lecture, Ecriture et Affectation

Exemple en langage C : (à exécuter dans les séances des TPs)

```
#include<stdio.h>
int main(){
int x,y,z;
printf("Donnez x: ");
scanf("%d",&x);
printf("Donnez y: ");
scanf("%d",&y);
z=x+y;
printf("la somme de x et y est :%d ",z);
 return 0;
}
```

Résultats :
Donnez x: 5
Donnez y: 7
la somme de x et y est :12

2.5.1.b Action d'Ecriture :

Syntaxe en langage algorithmique : ***Ecrire*(‘Message texte’, Variable1, Variable2, …)**

Syntaxe en langage C : ***printf*(‘Message texte %FORMAT1 %FORMAT2’, variable1, variable2,…) ;**

FORMAT1, FORMAT2, etc., sont des descripteurs de formatages des données en sortie.

Nous rapportons les formatages les plus courants dans la Table suivante :

Format	Explication
%c	Caractère
%s	String (chaîne de caractères)
%d	Int (entier)
%f	Float (réel)

Exemple : (à exécuter aux séances de TPs)

```
#include <stdio.h>
 int main()
{
   char code;
   int i=5;
   float y = 3.5;
   char nom[]="Skikda University Informatique 2024/2025";
   printf("Tapez un caractère :");
   scanf("%c", &code);
   printf("%c\n", code);
   printf("i = %d\n",i);
   printf("y = %f\n",y);
   printf("Nom =%s\n",nom);

   return 0;
}
```

Exécution :

```
Tapez un caractère :J
J
i = 5
y = 3.500000
Nom =Skikda University Informatique 2024/2025
```

2.5.1.c Action d'affectation :

Syntaxe en langage algorithmique :

vari ← valeur (vari étant une variable et valeur une valeur simple ou expression)

(vari et valeur sont de **même type**, ou, il y aura **conversion de type**, ou **erreur** …).

Exemples : x ← 12

y ←2*x-3

Syntaxe en langage C :

Vari = valeur ;

Où :

Vari est une variable d'un type élémentaire : logique, char, entier, réel, string.

(langage C : boolean, char, int, float, char [])

Exemples :

X=12;

y = 2*x – 3;

2.5.2 ***Les actions de contrôle*** : Ces actions n'agissent pas sur les valeurs des variables, mais agissent sur la valeur du compteur ordinal (**CO** : *registre contenant le numéro de l'action à exécuter*). Nous avons alors trois types d'actions de contrôle :

2.5.2.a ***La séquence (naturelle)*** : Cette action survient suite à l'exécution d'une action simple.

Effet : **CO ← CO + 1**. (Action suivante)

A noter que cette action est effectuée automatiquement (automatisme de la machine).

Exemples : On se réfère aux exemples précédents de ce chapitre.

Dans ces exemples, toutes les actions sont exécutées de façon séquentielle (aucune rupture de séquence : Ni retour en arrière, ni saut en avant).

2.5.2.b *Rupture de la séquence naturelle par saut en avant* : Cette action survient suite à l'exécution d'une action d'alternative (Ces actions seront développées plus loin dans ce cours).

Effet : **CO ← CO + *Déplacement***.

Ici, **déplacement** est le nombre d'actions à '***sauter'*** pour arriver à l'action désirée. Ce saut est effectué explicitement par programmation.

2.5.2.c *La rupture de séquence naturelle par saut en arrière* : Cette action est effectuée suite à l'exécution d'une action de répétition (boucle). Ces actions seront développées plus loin dans ce cours.

Effet : **CO ← CO – déplacement**.

Déplacement, ici, est le nombre d'actions à sauter en arrière pour retourner à un point désiré du programme.

Les actions *d'alternative et les actions de répétition* seront présentés dans les chapitres suivants (chapitre 3 et chapitre 4, respectivement).

2.6. Construction d'algorithmes simples

Dans cette section, nous construisons deux algorithmes simples, que nous traduirons par la suite en langage C.

Cas 1 : (à refaire aux TDs et TPs)
Nous voulons déterminer l'indice de masse corporelle : IMC, d'une personne.
On donne :

IMC = poids (en kg) / taille² (en m)

1. Analysez le problème
2. Ecrire un algorithme qui calcule l'IMC
3. Traduire l'algorithme en langage C.

Solution :

1. **Analyse du problème** :
   Il s'agit de déterminer les Données et les résultats attendus du problème.
   Ici :
   Données : poids (réel), taille (réel)
   Résultats : IMC : (réel) = poids/ taille²

2. **Algorithme** :

```
Algorithme Calcul_IMC
Données : p,t : réel ;
Résultats : IMC ;
Début
        Ecrire(‘ Donnez le poids :’) ;
        Lire(p)
        Ecrire(‘ Donnez la taille:’) ;
        Lire(t)
        IMC= p/(t*t)
        Ecrire(‘ IMC =’,IMC) ;
Fin
```

3. **Programme en C :**

```
#include <stdio.h>
#include <stdlib.h>
int main() {
float poids;
float taille;
float taille2;
float IMC;
// On demande à l'utilisiateur les deux nombres
printf("Calcul de l'IMC : ");
printf ("\n Entrez votre taille en m : ");
scanf ("%f", &taille);
printf ("\n Entrez votre poids en kg :");
scanf ("%f",&poids);

taille2=taille*taille;
IMC = poids/taille2;
 printf("\n IMC = %5.2f", IMC);              // Format de sortie du IMC
}
```

Résultats :

```
Calcul de l'IMC :
 Entrez votre taille en m : 1.75

 Entrez votre poids en kg :85

 IMC = 27.76
```

Cas 2 :

Effectuez la même démarche pour calculer les mesures de distance suivantes :

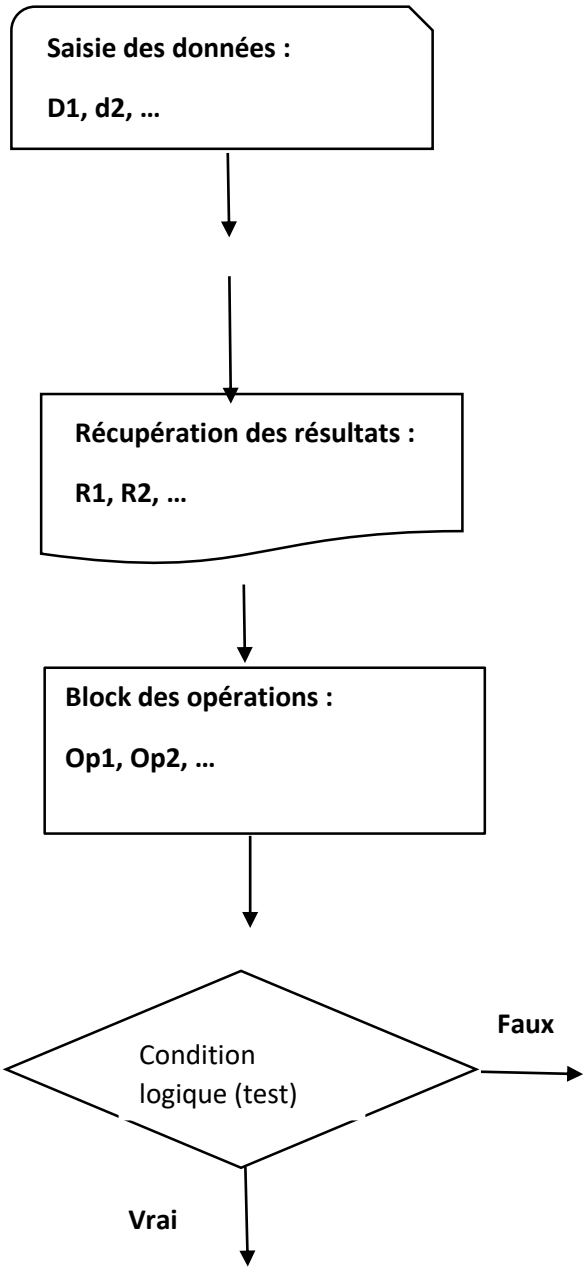
- La distance dite de la norme 1, définie par :d1(M1,M2)=|x1−x2|+|y1−y2|
  Où M1(x1,y1), M2(x2,y2) sont des points du plan cartésien.
- La *distance euclidienne* :d2(M1,M2)=$\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$

A corriger dans les TDs et TPs.

2.7. Représentation d'algorithmes par organigrammes

Les organigrammes sont des outils graphiques (visuels) pour représenter les algorithmes.

Voici les symboles utilisés par ce mode d'expression des algorithmes :



Nous réalisons alors ci-après l'organigramme de l'algorithme1, ci-haut.