1. INTRODUCTION

Le langage C a été mis au point au début des années 1970 et normalisé en 1988. C'est un langage de haut niveau qui génère un code très rapide grâce à un compilateur très performant.

2. STRUCTURE D'UN PROGRAMME C

Le langage **C** est un langage structuré, son programme de base est constitué de :

- Partie ent ête (Directive)
- Programme principal (Fonction principale)
- Partie traitement (Corps du programme = déclarations + instructions)

3. **EXEMPLE :** On essayera de faire la somme de deux nombres entiers lus.

Q1 : Ecriture l'algorithme correspondant ?

```
Algorithme somme
      Variables
          X, Y, Som : :entier ;
      DEBUT
        Ecrire ('entrer deux nombres entiers') ;
        Lire (x) ;
        Lire (y) ;
        Som← x+y ;
        Ecrire (Som) ;
       Fin.
```

Q2 : Ecriture son programme en C ?

```
      #include <stdio.h>  /* Partie entête ici C'est juste pour inclure des
bibliothèques */
       main()     /*Programma principale*/
      {  int X, Y, Som ;  /*Déclaration*/
      Printf(''entrer deux nombres entiers'') ; /* écriture d'un message */
      scanf("%i" , &X); /*Lecture de X*/
      scanf("%i" , &Y); /*Lecture de Y*/
      Som = X+Y ;  /*Calcul  la Somme*/
```

Remarque : La parenthèse ouvrante {joue le rôle de Début et la fermante} celui de Fin.

Explication de ce petit programme

- Ce programme principal `main()` utilise dans la partie traitement (corps du programme) deux différentes opérations `scanf()` et `printf()`, ce sont des opérations (fonctions) d'entrée/sortie (input/output),il utilise aussi une déclaration `int X,Y,Som` ; et une affectation `S = X+Y ;`
- Les opérations de déclaration et affectation sont reconnues par le compilateur par contre les opérations d'input/output ne le sont pas, donc il faut les inclure à l'aide de la directive `#include` qui précise au compilateur dans quel fichier se trouve la définition de ces fonctions.
- Que signifie `stdio.h` ?

  `Std : Standard`
  `io : input/output`
  `.h : header (entête)`
- L'inclusion qui est une opération appartenant aux instructions du pré processeur se fait comme suit : `#include <stdio.h>`

- Que signifie %i et & dans les opérations d'input/output `scanf()` et `printf()` ?
  `%i` : veut dire que c'est le format entier (i comme integer),
  `%i` est équivalent à `%d`
  `&` : cet opérateur doit précéder toute variable de type (`entier, réel, caractère …`).

REMARQUES

- Attention un mot écrit en minuscule est différent de celui qui est écrit en majuscule. Si vous écrivez `main()` et `Main()` ce n'est pas la même chose
- L'écriture du programme C doit être en minuscule !
- Le langage C permet la déclaration des variables à n'importe quel endroit dans le programme, il n'a pas une zone précise appelée zone déclarative. Il est permissif.
- Par contre la déclaration des constantes est dans l'entête (partie du pré compilateur), à l'aide de `#define`.
- En C, toute déclaration et/ou toute instruction se termine par un point- virgule ; sauf les instructions destinées au pré processeur comme par exemple `#include` …
- A chaque fois qu'on utilise une fonction standard, on doit inclure son fichier entête !

3. TYPES DE DONNEES

- Pour l'instant, on va étudier seulement trois (03) types de données (Variables) : les entiers, les réels et les caractères.
- **`int`** pour les entiers (`integer`)
- **`float`** pour les réels (flottant ou virgule flottante)
- **`char`** pour les caractères

4 .COMMENT DECLARER UNE VARIABLE ?

Algorithme	**Langage C**
Déclaration	Pas d'équivalent
Variables	Pas d'équivalent
A : entier ;	int A ;
C, D : réel ;	float C, D ;
K : caractère ;	char K ;

Aussi la déclaration d'une variable en C, se fait dans le corps du programme et non dans une zone déclarative comme dans un algorithme.

5. COMMENT DECLARER UNE CONSTANTE ?

Elles sont déclarées différemment des variables, on doit les définir dans la partie entête du programme C à l'aide de la directive #define et de la façon suivante :

`#define nom_de_la_constante valeur_de_la_constante`

Algorithme	Langage C
Constante	Pas d'équivalent
PI = 3.14 ;	#define PI 3.14
Déclaration	Pas d'équivalent

6. FICHIERS ENTETES

- On a déjà vu ce genre de déclaration quand on a écrit notre premier programme en langage C qui fait la somme de deux entiers X et Y dans Som, on a utilisé :

#include <stdio.h>

- D'une manière générale cela s'écrie comme suit :

#include <nom_du_fichier_entête>

- **Attention** : On n'inclut pas seulement les fichiers input /output mais il y en a d'autres qu'on verra par la suite !

7. LES OPERATIONS ELEMENTAIRES DU LANGAGE C

- **L'affectation :** c'est l'attribution d'une valeur ou d'un résultat d'une opération à une variable de même type.

Algorithme	Langage C
Début ……. ……. X ← 5 ; Y ← 18 ; SOM ← X+Y ; …. ….. FIN	X = 5 ; Y = 18 ; SOM = X+Y ; …… ……

- **La lecture (input)** : C'est l'attribution d'une valeur choisie par l'utilisateur à une variable d'entrée. La valeur choisie doit être de même type que la variable d'entrée. Cette opération se fait à l'aide de la fonction scanf() ; En générale cette fonction s'écrit sous la forme suivante :

scanf('' indicateur_de_type '', &nom_de_la_variable) ;

L'opérateur & précède toute variable de type scalaire (entier, réel ou caractère).

Algorithme	Langage C
Lire(X) ;	scanf('' %d '', &X) ;

NB : la variable x ici est de type entier c pour ça %d

- **L'écriture (output)** : C'est l'affichage (l'impression) du contenu d'une variable, d'une constante ou d'un simple message. Cette opération se fait à l'aide de la fonction `printf()` ; En générale cette fonction s'écrit sous la forme suivante : `pintf(''` indicateur_de_type `'',` nom_de_la_variable) ; ou bien `printf('' message '');`
-

Algorithme
Ecrire ('Bonjour') ; Ecrire(S) ;

Langage C
printf('' Bonjour '') ; printf('' %i '', Som);

`%i` est un indicateur de format (i comme **`integer`** c'est-à-dire entier). Vous aurez la liste de tous les indicateurs de format par la suite. Vous avez remarqué que le message doit être mis entre deux (02) double côtes `''message''`.

- **`L'incrémentation/la décrémentation`** : C'est une affectation un peu particulière, car on trouve le nom de la variable de part et d'autre du symbole d'affectation (← en algorithme et = en langage C).
  - L'incrémentation permet d'augmenter d'une valeur une variable par contre la décrémentation permet de diminuer d'une valeur une variable. Cette valeur est appelée le pas et le pas peut être positif ou négatif.

Algorithme	**Langage C**
Début	
X ← X + 1 ;	X = X + 1 ;
Y ← Y − 1 ;	Y = Y − 1 ;
A ← A + 2 ;	A = A + 2 ;
B ← B − 5 ;	B = B − 5 ;
Fin	

Attention : Il y a une écriture en langage C que je vous conseille d'éviter quoiqu'elle soit permise. `X = X + 1 ;` peut s'écrire `X + + ;` `Y = Y − 1` ; peut s'écrire `Y − − ;`

8. LES OPERATIONS ARITHMETIQUES

- L'addition +
- La soustraction –
- La multiplication *
- La division /
- Le modulo % c'est le reste de la division Euclidienne.

Algorithme	**Langage C**
Début	
Delta ← b² - 4ac ;	Delta = b*b − 4*a*c ;
Q ← X/2;	Q = X / 2;
R ← X − 2Q;	R = X − 2*Q;
Fin	

9. LES OPERATEURS LOGIQUES

Une condition peut être elle-même composée de plusieurs conditions reliées entre elles par des opérateurs logiques tels que et, ou, non. En langage C :

- Le et est représenté par &&
- Le ou est représenté par ||
- Le non est représenté par !

Algorithme	Langage C
A et B	A && B
X OU Y	X || Y
Non Z	! Z

10. LES OPERATEURS DE COMPARAISON :

On les appelle aussi signes de test :

==	teste si les deux informations sont égales.
<	teste si l'information à gauche est inférieure à celle de droite.
>	teste si l'information à gauche est supérieure à celle de droite.
<=	teste si l'information à gauche est inférieure ou égale à celle de droite.
>=	teste si l'information à gauche est supérieure ou égale à celle de droite.
!=	teste si les deux informations sont différentes.

Exemple : On va écrire deux instructions presque identiques pour ordonner la machine d'afficher le contenu d'une variable S de type entier.

- `printf(''%i'', S) ;`
- `printf(''%i\n'', S) ;`

La première instruction ordonne la machine à écrire la variable S et de rester sur la même ligne.

La seconde instruction ordonne la machine à écrire la variable S et d'aller à la ligne suivante, c'est ce qu'on appelle un retour chariot et cela grâce à \n.

NB : l'intérêt de cet exemple à cet endroit est de comprendre l'option \n

11. LES COMMENTAIRES

Pour rendre notre programme plus lisible, on doit le commenter c'est-à-dire écrire des commentaires pour expliquer ce qu'on est en train de faire. Ces commentaires seront ignorés par le compilateur C.

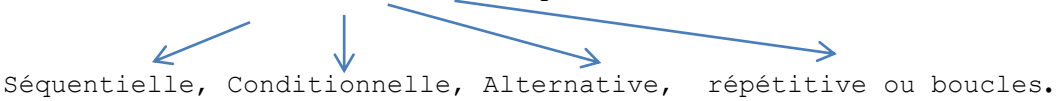
- Tout commentaire est mis entre /* et */.
- Exemple : /* ceci est un commentaire* /

12. LES STRUCTURES DE CONTROLE

On distingue deux types de structures de contrôle:

- **les conditions** :elles permettent d'écrire dans le programme des règles comme « Si ceci arrive... alors fais cela ...» ;
- **Les boucles**: elles permettent de répéter plusieurs fois une série d'instructions.
- On les appelle aussi les primitives, elles servent à contrôler le déroulement d'un traitement.

Les instructions : peuvent s'exécuter de différentes manières :



12.a Le traitement séquentiel : C'est une suite d'instructions qui s'exécutent l'une à la suite de l'autre sans aucune contrainte.

Algorithme

```
Début
   instruction1 ;
   instruction2 ;
  …
  Instruction n ;
Fin
```

Langage C

```
Instruction1 ;
Instruction2 ;
…
 instruction n;
```

12.b Le traitement conditionnel : Dans ce cas l'instruction est exécutée que si la condition est vérifiée. On dit que l'instruction est sous condition.

```
Algorithme

Début

   Si condition(s) alors

      Action ;

Finsi

fin
```

```
Langage C

 if condition(s)

   action ;
```

NB : on remarque que le mot alors en algorithme n'a pas d'équivalent en langage C de même pour le vocable finsi.

12.c Le traitement alternatif (si …sinon…..): Comme son nom l'indique, si ce n'est pas l'une ça sera l'autre ; c'est-à-dire si la condition est vérifiée l'action 1 est exécutée si ce n'est pas le cas l'action 2 sera exécutée et en aucun cas les actions 1 et 2 ne seront exécutées en même temps !

Algorithme

```
Début
…
 Si condition(s) alors
     Inst1 ;
     Inst2 ;
     Inst3 :
     ….
     .. ;.
     Instn :

Sinon
     Inst1 ;
     Inst2 ;
     Inst3 :
     ….
     .. ;.
     Instn :
finsi
     …
fin
```

Langage C

```
       …
 if condition(s)
    Inst1 ;
    Inst2 ;
    Inst3 :
    ….
    .. ;.
    Instn
else
    Inst1 ;
    Inst2 ;
    Inst3 :
    ….
    .. ;.
    Instn :
```

NB : si on a plusieurs actions que ce soit dans le alors ou/et dans le sinon, on utilisera des blocs. Début et fin en algorithme et les parenthèses ouvrantes et fermantes pour le langage C.

12 .d L'instruction selon le cas : Elle indique le choix multiple, et au lieu d'utiliser une cascade de si alors sinon, on préfère cette primitive pour nous faciliter l'écriture de l'algorithme ou le programme et le rendre plus lisible.(Voir cours sur algorithme).

Algorithme	**Langage C**
Début … … **Selon le cas** variable Cas1 : action1(s) ; Cas2 : action2(s) ; …….. Casn : actionn(s) ; Finselon ….. FIN	**switch** variable_de_choix case choix1 :action ;breack ; case Choix2 :action2 ; ;breack ; … case choix n :action n ;breack ;

Il existe un autre format d'écriture qui utilise le mot « autre » en algorithme qui sera traduit par « default » en langage C.

Algorithme	**Langage C**
Début Selon le cas variable_de choix Choix1 : action1 ; Choix2 :action2 ; choix n :action n ; … sinon : action n+1 ; Finselon … Fin	case switch variable_de_choix case choix1 :action1 ;breack ; case choix2 :action2 ;breack ; … Case Choix n :action n ; breack ; default : action n+1 ; breack ;

NB : En langage C la primitive switch utilise un bloc { } qui englobe tous les cas possibles et à la fin de chaque action il y a l'instruction breack pour casser la séquence. La finselon de l'algorithme n'a pas d'équivalent en C. **Exemple sur le SWITCH :**

```
#include <stdio.h>
main(s)
{   int M ;
printf(''Entrer une valeur entière'') ;
  scanf(''%d'', &M) ;
 switch (M)
{        case 1 : printf('' c'est Dimanche '') ;breack ;
             case 2 : printf(''c'est lundi'') ;breack ;
             case 3 : printf(''c'est Mardi'') ;breack ;
             case 4 : printf(''c'est mercredi'') ;breack ;
             case 5 : printf(''c'est jeudi'') ;breack ;
             case 6 : printf(''c'est vendredi'') ;breack ;
             case 7 : printf(''c'est Samedi'') ;breack ;
  default : printf(''Votre nombre ne correspond à aucun jour'');
 }  }
```

12.d Le traitement répetitif ,itératif ou boucles : On a trois primitives (instructions) pour contrôler un traitement répétitif.

A. Pour i allant de vi à vf (pas =p) faire
B. tant que (condition(s)) faire
C. répéter….jusqu'à (condition(s))

A. La boucle `Tant que faire` : Son principe est très simple :

- tant que la condition ou les conditions est/sont vérifiée(s), on exécute l'instruction ou les instructions qui est/sont à l'intérieur de la boucle.
- Une fois la condition ou les conditions n'est/sont plus vérifiée(s) on sort de la boucle. Si on a plus d'une action à l'intérieur de la boucle, on utilisera un bloc.

Algorithme

```
Début

Tantque   (condition(s)) Faire

    Action1 ;
    Action2 ;
    …….
    Action
Fintantque
 Fin
```

Langage C

```
while condition(s)
  action1 ;
  action2 ;
   …….
   Action ;
```

- On remarque que le mot clé Faire n'a pas d'équivalent en C, du même pour la fintantque.

Exemple : Afficher les dix (10) premiers nombres entiers positifs et chaque nombre est écrit sur une ligne.

```
#include <stdio.h>
main()
{  Int co ; /*déclaration du compteur*/
 co = 1 ;  /*initialisation du compteur*/
 while (co <= 10)   /*condition d'arrêt*/
    {   printf(''%d\n'', co) ;  /*affichage du compteur*/
        co = co+1 ;   /*incrémentation du compteur*/
}
```

B. La boucle `Pour` : Contrairement à la boucle tant que où devant le `while` on ne trouve que la/les condition(s) ; devant le `pour` (`for` en C) on trouve l'initialisation d'une variable, la condition d'arrêt et le pas (incrémentation ou décrémentation).

ALGORITHME

```
Pour V allant de Vi à Vf (pas = P) faire
    instruction(s) ;
Finpour
```

Langage C

```
for (initialisation ; condition ; pas ;)
action(s) ;
```

- Où V est le nom d'une variable déclarée.
- Vi est la valeur initiale (valeur de départ).
- Vf est la valeur finale (valeur d'arrive).

- P est le pas (incrémentation on dit que le pas est positif ou décrémentation on dit que le pas est négatif).
- Le même principe que la boucle tant que, si on a plus d'une action, un bloc est nécessaire.

Reprenons l'exemple traité avec la boucle while et écrivons le en utilisant la boucle for.

```
#include <stdio.h>
main()
 {
 int co ;
 for (co = 1 ; co <= 10 ; co = co+1 ;)
        printf(''%d\n'', co);
}
```

Faisons un parallèle entre les deux écritures **(while et for).**

```
       Boucle while
#include <stdio.h>
main()
{     int co;
Co = 1;
while (co <= 10)
{
   printf(''%d\n'', co);
  co=co+1;
    }  }
```

```
           Boucle for
#include <stdio.h>
main()
{ int co;
for (co=1; co<=10; co=co+1;)
printf(''%d\n'', co);

}
```

- On Remarque qu'avec la boucle for on a gagné deux (02) lignes de code: l'initialisation (`co=1;`) et l'incrémentation (`co=co+1;`) et même un bloc { }.
- Par conséquent si on a la possibilité d'utiliser la boucle for n'hésiter pas un seul instant ! à condition qu'on ait une et une seule variable V, une valeur initiale Vi, une valeur de fin Vf et le pas P (positif : incrémentation ou négatif : décrémentation).

C .La boucle tant que condition faire: Cette boucle est un peu spéciale, on traite l'action puis on vérifie la condition. C'est pour cela que cette boucle n'est utilisée que si on est assuré qu'elle va s'exécuter au moins une fois. Bizarrement, elle ressemble à la boucle (répéter jusqu'à) qu'on a étudié en algorithmique. Sa syntaxe est la suivante :

```
     Algorithme
Faire
Action (s);
Tantque
condition(s);
```

```
     Langage C
do
action(s) ;
while (condition(s)) ;
```

Nb *: Traitons le même exemple du comptage avec la boucle (do while).*

```
#include <stdio.h>
main()
{   int co ;  co = 1 ;
   do
          {    printf(‘’ %d\n’’, co) ;
              co = co+1;          }
   while (co <= 10);    }
```

Remarque:

- la priorité est donnée à la boucle (`for`) puis la boucle (`while`) et en dernier la boucle (`do while`) car son problème est qu'il faut être assuré qu'elle va s'exécuter au moins une fois !
- La fonction printf est une fonction d'impression formatée, ce qui signifie que les données sont converties selon le format particulier choisi. Ci-dessous le tableau des formats d'impression pour la fonction printf

Format	Conversion	en Ecriture
%s	char*	chaine de caractères
%d	int	décimale signée
%u	unsigned int	décimale non signée
%o	unsigned int	octale non signée
%c	unsigned char	Caractère
%e	double décimale	notation exponentielle
%x	unsigned int	hexadécimale non signée

- Les opérateurs d'incrémentation ++ et de décrémentation -- s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i). Dans les deux cas la variable i sera incrémentée, toutefois dans la notation suffixe la valeur retournée sera l'ancienne valeur de i alors que dans la notation préfixe se sera la nouvelle
- Le langage C ne dispose que de la notation / pour désigner à la fois la division entière et la division entre flottants. Si les deux opérandes sont de type entier, l'opérateur / produira une division entière (quotient de la division). Par contre, il délivrera une valeur flottante dès que l'un des opérandes est un flottant.
- L'opérateur % calcule le reste de la division Euclidienne (modulo). Il ne s'applique qu'à des opérandes de type entier.
- Le tableau suivant classe les opérateurs par ordres de priorité décroissants, les opérateurs sur une même ligne ayant une priorité égale (on évalue alors de gauche à droite).

Priorité 1 -	( ) [ ]
Priorité 2 (unaire)	-- ++ !
Priorité 3	* / %
Priorité 4 (binaire)	+ -
Priorité 5	< <= > >=
Priorité 6	== !=
Priorité 7	&&
Priorité 8	||
Priorité 9	%= /= *= -= += =