Partie II : Plus sur le langage C

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
- la virgule flottante est composée d'une mantisse et d'un exposant. Par exemple la valeur 3.14 peut s'écrire sous un autre format $314 * 10^{-2}$: ici la mantisse est 314 et l'exposant est -2.

COMMENT DECLARER UNE VARIABLE ?

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

NB : Dans la partie III de ce fascicule vous aurez un tableau récapitulatif de tous les types en langage C.

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 !

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**
Déclaration	Pas d'équivalent
Constante	Pas d'équivalent
PI = 3.14 ;	#define PI 3.14

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 A et B dans S, 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 !

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	{
…	…
A ← 5 ;	A = 5 ;
B ←18 ;	B = 18 ;
S ← A + B ;	S = A + B ;
…	…
Fin.	}

- 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**
Début	{
…	…
Lire(A) ;	scanf('' %d '', &A) ;
…	…
Fin.	}

- 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	**Langage C**
Début	{
…	…
Ecrire('Bonjour') ;	printf(‘’ Bonjour ‘’) ;
Ecrire(S) ;	printf(‘’ %i ‘’, S) ;
…	…
Fin.	}

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

- L'incrémentation ou 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 est permise. C'est ce genre de fantaisies qui rend le code du langage C illisible ! Malheureusement il y a ceux qui en raffolent.

X = X + 1 ; peut s'écrire X + + ;

Y = Y – 1 ; peut s'écrire Y - - ;

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	{
…	…
Δ ← b² - 4ac ;	Delta = b*b – 4*a*c ;
Q ← X/2;	
R ← X – 2Q;	R = X % 2;
…	…
Fin.	}

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

SIGNES DE COMPARAISON

On les appelle aussi signes de test.

== teste si les deux informations sont égales. Ne pas confondre avec l'affectation.

< teste si l'information à gauche est inférieur à celle de droite.

< teste si l'information à gauche est supérieur à celle de droite.

<= teste si l'information à gauche est inférieur ou égale à celle de droite.

>= teste si l'information à gauche est supérieur ou égale à celle de droite.

!= teste si les deux informations sont différentes.

CARACTERE D'ECHAPPEMENT

- 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.

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* /

LES STRUCTURES DE CONTROLE

- On les appelle aussi les primitives, elles servent à contrôler le déroulement d'un traitement.
- Un traitement peut s'exécuter de différentes manières :
  - Séquentielle.
  - Conditionnelle.
  - Alternative.
  - Itérative (ou répétitive).

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

Algorithme	**Langage C**
Début	{
Instruction1 ;	instruction1 ;
Instruction2 ;	instruction2 ;
…	…
Instruction n ;	instruction n ;
Fin.	}

- **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	**Langage C**
(* Première écriture*)	/* Première écriture*/
Début	{
…	…
Si condition(s) alors	if condition(s)
Action ;	action ;
Finsi	
…	…
Fin.	}

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

Algorithme	**Langage C**
(* Deuxième écriture*)	/* Deuxième écriture*/
Début	{
…	…
Si condition(s) alors	if condition(s)
Début	{
Action1 ;	action1 ;
Action2 ;	action2 ;
Fin	}
Fin si	…
Fin	}

- **Le traitement alternatif :** 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	**Langage C**
(*Premier format*)	/*Premier format*/
Début	{
…	…
Si condition(s) alors	if condition(s)
Action1 ;	action1 ;
Sinon	else
Action2 ;	action2 ;
Finsi	
…	…
Fin.	}

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. C'est l'autre format d'écriture.

Algorithme

```
(*Deuxième format*)
Début
    …
    Si condition(s) alors
        Début
            Action11 ;
            Action12 ;
            Action13 ;
        Fin
    Sinon
        Début
            Action21 ;
            Action22 ;
        Fin
    Fin si
Fin.
```

Langage C

```
/*Deuxième format*/
{
    …
    if condition(s)
        {
            action11 ;
            action12 ;
            action13 ;
        }
    else
        {
            action21 ;
            action22 ;
        }
    …
}
```

- **La primitive selon :** 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.

Algorithme

```
Début
    …
    Selon variable_de_choix

        Choix1 : action1 ;
        Choix2 :action2 ;
        …
        Choix n :action n ;
    Finselon
    …
Fin
```

Langage C

```
{
    …
    switch variable_de_choix
    {
       case choix1 :action1 ;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

```
Début
    …
    Selon variable_de_choix
            Choix1 : action1 ;
            Choix2 :action2 ;
            …
            Choix n :action n ;
            Autre : action n+1 ;
    Finselon
    …
Fin
```

Langage C

```
{
    …
    switch variable_de_choix
    {
        case choix1 :action1 ;breack ;
        case choix2 :action2 ;breack ;
        …
        case choix n :action n ;breack ;
        default : action n+1 ;
    }
    …
}
```

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 DE BILAN SUR LE SWITCH

```
#include <stdio.h>
main()
{
        int M ;
        printf(''Entrer une valeur entière'') ;
        scanf(''%d'', &M) ;
        switch (M)
                {
                        case 1 : printf('' c'est Janvier '') ;breack ;
                        case 2 : printf(''c'est Février'') ;breack ;
                        case 3 : printf(''c'est Mars'') ;breack ;
                        case 4 : printf(''c'est Avril'') ;breack ;
                        case 5 : printf(''c'est Mai'') ;breack ;
                        case 6 : printf(''c'est Juin'') ;breack ;
                        case 7 : printf(''c'est Juillet'') ;breack ;
                        case 8 : printf(''c'est Aout'') ;breack ;
                        case 9 : printf(''c'est Septembre'') ;breack ;
                        case 10 : printf(''c'est Octobre'') ;breack ;
                        case 11 : printf(''c'est Novembre'') ;breack ;
                        case 12 : printf(''c'est Décembre'') ;breack ;
                        default : printf(''Votre nombre ne correspond à aucun mois'');
                }
}
```

- **Le traitement itératif :** On dispose de trois primitives pour contrôler un traitement répétitif.
  - La boucle tant que faire
  - La boucle pour faire
  - La boucle faire tant que (c'est l'équivalente de la boucle **répéter**).
- **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	**Langage C**
Tantque condition(s) Faire	while condition(s)
Action ;	action ;
Fintantque	

Algorithme	**Langage C**
Tantque condition(s) Faire	while condition(s)
Début	{
Action1 ;	action1 ;
Action2 ;	action2 ;
Fin	}
Fintantque	

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*/
           }
}
```

- **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	**Langage C**
Pour V allant de V_I à V_F pas P faire	for (initialisation ; condition ; pas ;)
Action ;	action ;
Finpour	

Le même principe que la boucle tant que, si on a plus d'une action, un bloc est nécessaire.

Algorithme	**Langage C**
Pour V allant de V_I à V_F pas P faire	for (initialisation ; condition ; pas ;)
Début	{
Action1 ;	action1 ;
Action2 ;	action2
Action3 ;	action3
Fin	}
Finpour	

Où V est le nom d'une variable déclarée.

V_I est la valeur initiale (valeur de départ).

V_F 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).

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 parallele 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 V_I, une valeur de fin V_F et le pas P (positif : incrémentation ou négatif : décrémentation).

- **La boucle faire tant que :** 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	**Langage C**
Faire	do
Action ;	action ;
Tantque condition(s) ;	while (condition(s)) ;

Ou bien l'autre format d'écriture:

Algorithme	**langage C**
Faire	do
Début	{
Action1 ;	action1 ;
Action2 ;	action2 ;
Action3 ;	action3 ;
Fin	}
Tantque condition(s) ;	while (condition(s)) ;

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:

Si j’ai à choisir entre ces trois (03) boucles, 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 !

BIBLIOGRAPHIE

1- Langage C : Cours et références
Pierre NERZIC Mars 2003

2- Langage C : Support de cours
Patrick CORDE Mars 2006

3- Programmation en langage C
M.C. BELAID

4- Algorithmique, structures de données et langage C
J.M. ENJALBERT Janvier 2005