De l'Algorithmique au Langage C
PARTIE 1

1. Introduction

Le codage d'un algorithme en langage C consiste à utiliser ce dernier pour écrire un programme correspondant exactement à l'algorithme à coder. Cette opération est assez facile si l'algorithme respecte les conventions étudiées précédemment.

2. Structure d'un programme en langage C

L'algorithmique et le langage C sont tous deux des langages structurés. Ils sont donc sur ce plan très voisins.
Le bloc de programme de l'algorithmique se traduit immédiatement en C par une paire d'accolades.

Algorithme	Langage C
DEBUT	{
FIN	}

Comme en algorithmique, des paires d'accolades peuvent être imbriquées, mais ne se chevauchent jamais.
Les expressions et les instructions sont définies comme en algorithmique :

- Une expression est un ensemble d'identificateurs et d'opérateurs formant une phrase évaluable.
- ***Une expression suivie d'un point-virgule est une instruction*** et les opérateurs classiques de l'algorithmique existent tous en langage C.

Et enfin, le langage s'exécute séquentiellement, dans l'ordre où sont écrites les instructions.

Algorithme	Langage C
PROGRAMME nomprogramme	[*directives au préprocesseur*]
Déclarations de variables globales	*Déclarations de variables globales*
	nom_du_programme
DEBUT	**{**
Déclarations de variables locales	*Déclarations de variables locales*
Instructions du programme	*Instructions du programme*
FIN	**}**

Un programme en langage C est constitué d'une ou plusieurs fonctions. Cette fonction s'appelle **main**. C'est un nom de fonction particulier qui signifie « principal ». **main** est la fonction principale de votre programme, **c'est toujours par la fonction main que le programme en C commence**.

Algorithme	Langage C
PROGRAMME nomprogramme	[*directives au préprocesseur*]
Déclarations de variables globales	*Déclarations de variables globales*
	int main()
DEBUT	{
Déclarations de variables locales	*Déclarations de variables locales*
Instructions du programme	*Instructions du programme*
	return 0 ;
FIN	}

Le **préprocesseur** ou pré-compilateur est un utilitaire qui traite le fichier source avant le compilateur. Le mot **include** en anglais signifie « **inclure** » en français. Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation. Ces fichiers existent déjà, des fichiers source tout prêts. On les appelle des bibliothèques ou aussi de librairies.
Les directives du préprocesseur commencent par **#,** et se terminent par un retour à la ligne **et pas de point-virgule (;**)
Exemple

```
#include <stdio.h> /*déclare les fonctions qui gèrent les entrées sorties, …*/
#include <stdlib.h> /*déclare les fonctions qui effectuent les conversions des nombres, …*/
#include <math.h> /*déclare les fonctions mathématiques de base, …*/

int main()
{

Déclarations de variables locales

 Instructions du programme
return 0 ;

}
```

La ligne **return 0** indique qu'on arrive à la fin de la fonction main et demande de renvoyer la valeur 0. En pratique, **0** signifie « tout s'est bien passé » et n'importe quelle autre valeur signifie « erreur ».

3. Les données d'un programme en langage C

Les types de base du langage C sont les caractères (**char**), les entiers (**int**) et les nombres en virgule flottante ou réel (**float**), ainsi que les tableaux.

Algorithme	Langage C
Caractère	char
Entier	int
Réel	float

Le type chaine de caractères n'existe pas en langage C. Il faut savoir que le langage c, n'est pas prédisposé à manipuler les chaînes de caractère.
Pour le langage c, une chaîne de caractère est un tableau à une seule colonne, rempli de caractères : une chaîne de caractères est une suite de caractères (char), le tout étant terminé par un caractère supplémentaire de code « **null** », soit en **ASCII à 00**. Cela permet simplement à votre ordinateur de savoir quand s'arrête la chaîne ! donc **une chaîne de caractères n'est rien d'autre qu'un tableau de type char**. Par exemple, l'expression **char prenom[7]** désigne un tableau dont l'identificateur est prenom et dont la taille est (6 + 1).

S	A	M	U	E	L	

Ce tableau pourra contenir un prénom de 6 caractères au maximum.

4. Déclaration des variables

Une variable doit être définie par le programmateur dans une déclaration, où l'on indique le nom que l'on désire lui donner, son type (char, int, float, ...) pour que le compilateur sache combien de mémoire il doit lui réserver et les opérateurs qui peuvent lui être associés, mais aussi comment elle doit être gérée (visibilité, durée de vie, ...). Les variables sont définies par ***le type suivi de l'identificateur***.
Exemple
int nombre **;**
float resultat **;**
float x,y **;**
Pour déclarer des variables de type chaîne de caractères, rappelez-vous qu'avec des chaînes de caractères, on travaille avec un tableau à une seule colonne (de façon simplifiée). On va donc devoir déclarer un tableau, qui devra être :
- **nommer**
- **dimensionner**
- **typer**

On aura une instruction du genre :
type nom_tableau [nombre_caractères_maximum+1]

Exemple :

char nomEleve[7]

5. Déclaration des constantes

La déclaration d'une constante reprend la forme utilisée pour une variable précédée du mot clé **const** et suivie, bien sûr, de la valeur de cette constante.

const type identificateur = valeur;

Voici un tableau de quelques exemples de déclaration de variables et de constantes :

Algorithme	Langage C
reponse : **Caractere** ;	**char** reponse ;
nom[0..6] : **tableau de caractère** ;	**char** nom[7];
x : **entier** ;	**int** x ;
delta : **réel** ;	**float** delta ;
const pi = 3.14 ;	**const float** pi = 3.14 ;

6. Les mots réservés

Un certain nombre de mots, appelés *mots-clefs*, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L'ANSI C compte 32 mots clefs :

auto	const	double	float	int	short	struct	unsigned
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static	union	while

7. L'affectation

En C, l'affectation est un opérateur à part entière. Elle est symbolisée par le signe =. Sa syntaxe est la suivante :

variable = expression ;

Le terme de gauche de l'affectation peut être une variable simple, un élément de tableau, mais pas une constante. Cette opération a pour effet d'évaluer *expression* et d'affecter la valeur obtenue à la *variable*.

8. Les opérateurs arithmétiques

Les opérateurs arithmétiques classiques sont l'opérateur unaire - (changement de signe) ainsi que les opérateurs binaires :

+ addition
- soustraction
***** multiplication
/ division
% reste de la division (modulo)

Ces opérateurs agissent de la façon attendue sur les entiers comme sur les flottants. Leurs seules spécificités sont les suivantes : contrairement à d'autres langages, le C ne dispose que de la notation / pour désigner à la fois la division entière (DIV) et la division entre flottants (Division réelle). 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.

Par exemple,

- float x;
- x = 3 / 2; //affecte à x la valeur 1.0 car 3 et 2 sont des entiers

  Par contre

  x = 3 / 2.; //affecte à x la valeur 1.5 car 3 est un entier et 2. est un réel

- L'opérateur % ne s'applique qu'à des opérandes de type entier. Si l'un des deux opérandes est négatif, le signe du reste dépend de l'implémentation, mais il est en général le même que celui du dividende.
- ***Notons enfin qu'il n'y a pas en C d'opérateur effectuant l'élévation à la puissance***. De façon générale, il faut utiliser la fonction ***pow(x,y)*** de la librairie ***math.h*** pour calculer x^y.

Remarque
L'affectation effectue une *conversion de type implicite* : la valeur de l'expression (terme de droite) est convertie dans le type du terme de gauche. Par exemple, le programme suivant

```
int main()
{
  int i, j = 2;
  float x = 2.5;
  i = j + x;
  x = x + i;
}
```

affecte à x la valeur 6.5 (et non 7), car dans l'instruction i = j + x;, l'expression j + x a été convertie en entier.

9. Les opérateurs relationnels

> strictement supérieur
>= supérieur ou égal
< strictement inférieur
<= inférieur ou égal
== égal
!= différent

10. Les opérateurs logiques booléens

&& et logique
|| ou logique
! négation logique

Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un ***int*** qui vaut 1 si la condition est vraie et 0 sinon.

1. Les opérateurs d'affectation composée

Voici quelques opérateurs d'affectation composée

+= -= *= /= %=

Pour tout opérateur **op**, l'expression
expression1 = expression1 **op** expression2

est équivalente à

expression1 **op=** expression2

Toutefois, avec l'affection composée, expression1 n'est évaluée qu'une seule fois.
Exemple

Algorithme	Langage C
x ← x+ y ;	x = x + y ; est équivalente à x += y ;
x ← x - y ;	x = x- y ; ou x -= y ;
x ← x / y ;	x = x/y ; ou x /= y ;
x ← x % y ;	x = x%y ; ou x %= y ;

11. Les opérateurs d'incrémentation et de décrémentation

On doit souvent incrémenter ou décrémenter une variable. Le langage C offre 2 opérateurs (unaires) pour effectuer ces opérations :

- ++ : incrémentation de 1
- -- : décrémentation de 1

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.

Exemple 1 sur la pré-incrémentation/pré-décrémentation :

l'instruction : **a = ++b;** est équivalente à :

```
b = b + 1;
a = b;
```

La valeur de l'expression ++b est la valeur de b après incrémentation.

```
int a = 3, b;
b = ++a;      /* a et b valent 4 */
```

Exemple 2 sur la post-incrémentation/post-décrémentation :

l'instruction : **a = b++;** est équivalente à :

```
a = b;
b = b + 1;
```

La valeur de l'expression b++ est la valeur de b avant incrémentation.

Par exemple,

```
int a = 3, b;
b = a++;      /* b vaut 3 et a vaut 4 */
```

12. Les initialisations de variables.

Deux solutions sont possibles :

L'initialisation après déclaration.	L'initialisation lors de la déclaration.
/* Déclaration */ **int** i; **int main**() **{** /* Initialisation */ i=2; /* les autres instructions */ **}** **return 0 ;**	/* Déclaration et Initialisation */ **int** i=2; **int main**() **{** . /* les autres instructions */ **}** **return 0 ;** .

13. Les entrées sorties au clavier et à l'écran

Sur un ordinateur donné, les fonctions permettant d'accéder dans le programme à des valeurs issues du clavier ou à destination de l'écran sont traitées par des routines qui sont spécifiques à l'architecture de l'ordinateur et à son système d'exploitation.

Pour rendre ceci transparent, et donc rendre le code portable d'une machine à l'autre, le compilateur C est accompagné d'un ensemble de bibliothèques dans lesquelles le détail de ces fonctions est implémenté. Pour pouvoir utiliser ces fonctions, il suffit d'inclure le fichier d'en tête de la bibliothèque, fichier qui porte le même nom quelle que soit la machine considérée : **stdio.h** pour les entrées/sorties standards.

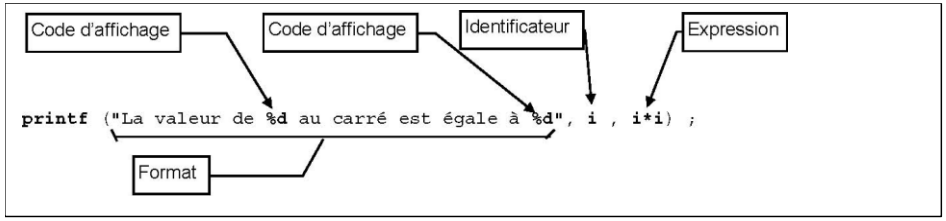
Ceci se fait en écrivant la directive de préprocesseur : **#include <stdio.h>** Au tout début du fichier contenant le programme en langage C. Cette précaution prise, la traduction de AFFICHER (ECRIRE) et (SAISIR (LIRE) est immédiate.

Les fonctions ***printf*** et ***scanf*** permettent d'afficher ou de lire simultanément plusieurs informations de type quelconque.

a. La fonction d'affichage.

Elle permet d'afficher des messages et/ou des valeurs de variables sous différents formats

```
Syntaxe:
printf(<"Format">,identificateur1, ....,identificateurn);
```



Les descripteurs de format ou code d'affichage sont décrits dans le tableau suivant :

Type	Format
Entier décimal	%d
Entier Octal	%o
Entier Hexadécimal (minuscules)	%x
Entier Hexadécimal (majuscules)	%X
Entier Non Signé	%u
Caractère	%c
Chaîne de caractères	%s
Flottant (réel)	%f
Scientifique	%e
Long Entier	%ld
Long entier non signé	%lu
Long flottant	%lf

b. La fonction de saisie.

Elle permet de saisir des valeurs de variables formatées à partir du clavier. Comme **printf** elle est composée d'un format et des identificateurs de variables à saisir. A la différence de **printf**, le format ne peut contenir de texte, il est juste composé du format des valeurs à saisir.

```
Syntaxe:
scanf(<"Format">,&identificateur1, ...., &identificateurn);
```

Exemple :
scanf ("%d", &x);
scanf ("%d %d ", &x, &y);
scanf ("%c %f ", &rep, &nombre);

Le symbole **&** est obligatoire devant les identificateurs car **scanf** attend des adresses et non des valeurs, **sauf devant un identificateur de chaîne de caractères.**

Voici quelques exemples d'algorithmes codés en C

Algorithme	Langage C

Programme saisie
DEBUT
Var x : entier ;
A : réel ;
SAISIR x ;
SAISIR A ;
FIN
	#include <stdio.h>
	int main()
	{
	int x;
	float A;
	scanf ("%d", &x);
	scanf ("%f", &A);
	return 0 ;
	}

Programme saisie
DEBUT
Var x : entier ;
A : réel ;
SAISIR x,y ;
FIN
	#include <stdio.h>
	int main()
	{
	int x;
	float A;
	scanf ("%d%f",&x,&A);
	return 0 ;
	}

Programme affichage
Var x : entier ;
DEBUT
x ← 0 ;
Afficher (vous avez le No ",x)
FIN
	#include <stdio.h>
	int main()
	{
	int x=0;
	printf ("vous avez le No %d",x) ;
	return 0 ;
	}

Programme affichage
Var age : entier ;
moy : réel ;
DEBUT
x ← 15 ;
moy ← 12,5 ;
Afficher("à ",age,"vous avez", moy) ;
FIN
	#include <stdio.h>
	int main()
	{
	int age=15;
	float moy=12.5;
	printf ("à %d vous avez %f",age,moy)
	return 0 ;
	}

c. Lecture et écriture des caractères

En C, Pour afficher un caractère sur l'écran, l'instruction (AFFICHER ou ECRIRE de l'algorithmique), est :

printf("%c", maLettre);

En C, Pour lire un caractère au clavier, l'instruction (SAISIR ou LIRE de l'algorithmique), on fait :

scanf("%c", &maLettre);

Et effectivement, c'est bien. **%c** indique que l'on attend un caractère, qu'on stockera dans maLettre (une variable de type char). Tout se passe très bien… tant qu'on ne refait pas un scanf.

En effet, vous pouvez tester les codes suivants :

```
#include <stdio.h>

int main( )

{
char maLettre ;

        printf("Entrer une lettre :") ;
        scanf("%c", &maLettre) ;
        printf("\nTres bien, vous avez entre %c", maLettre);

return 0;
}
```

Ou

```
#include <stdio.h>

        int main( )

        {
        char maLettre;
                printf("Entrer une lettre :");
                scanf("%c", &maLettre);
                printf("\nVous avez entré %c", maLettre);
                printf("\nEntrer une autre lettre :");
                scanf("%c", &maLettre);
                printf("\nVous avez entré cette fois %c", maLettre);
        return 0;
        }
```

Normalement, ce deuxième code est censé vous demander une lettre et vous l'afficher, et cela deux fois. A l'exécution, Vous entrez une lettre, d'accord, mais… le programme s'arrête de suite après, il ne vous demande pas la seconde lettre ! On dirait qu'il ignore le second scanf.

En fait, quand vous entrez du texte en console, tout ce que vous tapez est stocké quelque part en mémoire, *y compris l'appui sur la touche Entrée* (\n).
Ainsi, la première fois que vous entrez une lettre (par exemple C) puis que vous appuyez sur *Entrée*, c'est la lettre C qui est renvoyée par le scanf. Mais la seconde fois, scanf renvoie le \n correspondant à la touche *Entrée* que vous aviez pressée auparavant !

L'instruction scanf pose des problèmes à partir de la seconde exécution du programme.

Une des solutions possibles est que vous pouvez utiliser la fonction **getche( )** qui est une fonction de **stdlib.h**. L'expression **scanf ("%c", & maLettre)** est équivalente à **maLettre = getche( )**

```
#include <stdio.h>
#include <stdlib.h>

int main( )

{
char maLettre;
        printf("Entrer une lettre :");
        maLettre = getche( ) ;
        printf("\nVous avez entre %c", maLettre);
        printf("\nEntrer une autre lettre :");
        maLettre = getche( );

        printf("\nVous avez entre cette fois %c", maLettre);

return 0;

}
```

d. Lecture et écriture des chaînes de caractères

Le langage C offre plusieurs possibilités de lecture ou d'écriture de chaînes :

- **utilisation du code de format %s dans les fonctions printf et scanf :**

Pour faire un printf ou un scanf il faut utiliser le symbole **%s** (s comme *string)*, qui signifie « chaîne » en anglais. Dans les deux cas la fonction reçoit l'adresse du début du tableau "char" contenant la chaîne de caractère, cela à cause de son format **%s**.
Par exemple, soit le tableau char nom[15]; l'adresse du début du tableau est ***nom*** ou ***&nom[0]***.

Exemple
Pour saisir (lire) au clavier une chaine de caractère et l'enregistrer dans une variable de type tableau de char dont l'identificateur est nom par exemple, on fait :
scanf("%s", nom) ou **scanf("%s", &nom[0]**)

Pour afficher le contenu d'une variable de type tableau de char dont l'identificateur est nom par exemple, on fait :

printf("%s", nom) ou **printf("%s", &nom[0]**)

Testez l'exemple suivant :

```
#include <stdio.h>
int main()
{
char nom[20], prenom[20], ville[25] ;

        printf("quelle est votre ville : ") ;
        scanf("%s", ville);
        printf("donnez votre nom: ") ;
        scanf("%s", nom) ;
        printf("donnez votre prenom: ") ;
        scanf("%s", prenom) ;

        printf("bonjour cher %s %s qui habitez à %s ", prenom, nom, ville) ;

return 0 ;
}
```

Le format %s peut être complété, aussi bien pour *scanf* que pour printf, par un nombre entier qui indique le nombre maximal de caractère à lire ou à afficher. Ainsi,

scanf("%10s", nom) lit au plus 10 caractère et le range dans le tableau "nom" (les autres caractères saisis sont perdus).
De même,

printf("%**.**10s", nom) /*attention au **Point** avant le nombre 10*/ affiche 10 caractère au plus de la chaine de caractère rangée dans "nom".

N.B
Exécutez le programme précédent avec comme ville Abomey et une deuxième fois Abomey Calavi et comparez les résultats des exécutions.

Remarque

Le code %s de *scanf* interdit la lecture d'une chaîne contenant des espaces.

- **Utilisation de GETS ou PUTS**

On peut aussi saisir des caractères, en utilisant la commande GETS ou PUTS ; ces fonctions permettent de stocker les retours à la ligne, et les espaces, choses que la fonction SCANF ne permet pas.

Testez les codes suivants :

```
#include <stdio.h>
int main()
{
char nom[20], prenom[20], ville[25] ;

        printf("quelle est votre ville : ") ;
        gets(ville);
        printf("donnez votre nom: ") ;
        scanf("%s", nom) ;
        printf("donnez votre prenom: ") ;
        scanf("%s", prenom) ;

        printf("bonjour cher %s %s qui habitez %s ", prenom, nom, ville) ;

return 0 ;
}
```

```
#include <stdio.h>
int main()
{
char nom[20], prenom[20], ville[25] ;

        printf("quelle est votre ville : ") ;
        gets(ville);
        printf("donnez votre nom: ") ;
        scanf("%s", nom) ;
        printf("donnez votre prenom: ") ;
        scanf("%s", prenom) ;

        printf("bonjour cher %s %s qui habitez ", prenom, nom) ;
        puts(ville);

return 0 ;
}
```

On peut aussi mettre des codes de contrôles dans un programme C :

Code de contrôle	Signification
\n	Nouvelle ligne
\a	Bip code ascii 7
\r	Retour chariot
\b	Espace arrière
\t	Tabulation
\f	Saut de Page
\\	Antislash
\"	Guillemet
\'	Apostrophe
\'0'	Caractère nul
\0ddd	Valeur octale (ascii) ddd
\xdd	Valeur hexadécimale dd

Exemple

```
#include <stdio.h>
void main()
{
printf("Salut je suis:\n\t Le roi \n\t\t A bientôt\a");
}
```

L'exécution donnera

Salut je suis
Le roi
A bientôt *(Plus un Bip sonore)*

14. Les instructions de contrôle

a. L'alternative simple

- *Pour une seule instruction se traduit par*

if (Condition)
instruction ;

```
#include <stdio.h>
int main ()
{
int i, n;
printf ("Entrer un nombre positif:");
scanf("%d",&n);
  if(n > 0)
  printf ("%d est un nombre positif",n);
return 0;
}
```

- Pour plusieurs instructions se traduit par un bloc

if (Condition**)**
{
instruction 1 **;**
instruction 2 **;**
…
instruction n **;**
}

```
#include <stdio.h>
int main ()
{
int i, n;
printf ("Entrer un nombre positif:");
scanf("%d",&n);
  if(n > 0)
  {
   printf ("%d ",n);
   printf ("est un nombre positif");
  }
  return 0;
```

b. L'alternative complète

- Pour une seule instruction se traduit par

if (Condition)
une seule instruction ;
else
une seule autre instruction ;

```
#include <stdio.h>
int a,b;
int main()
{
/* Saisie de a et de b */
printf("Donnez les valeurs de a et de b ");
scanf("%d %d",&a,&b);
/* Structure SI ALORS SINON */
        if (a>b)
        printf("a=%d est supérieur à b=%d \n ",a,b);
        else
        printf("a=%d est inférieur ou égal à b=%d \n",a,b);
return 0 ;
}
```

- Pour plusieurs instructions (bloc d'instructions) se traduit :

if (Condition)
{
les instructions ;
}
else
{
les autres instruction ;
}

Exemple

```
#include <stdio.h>
int a,b;
int main()
{
/* Saisie de a et de b */
printf("Donnez les valeurs de a et de b ");
scanf("%d %d",&a,&b);
/* Structure SI ALORS SINON */
        if (a>b)
        {
        printf("a=%d est supérieur à b=%d",a,b);
        printf("\n");
        }
        else
        {
        printf("a=%d est inférieur ou égal à b=%d",a,b);
        printf("\n");
        }
return 0 ;
}
```

c. L'alternative imbriquée

```
#include <stdio.h>
int main ()
{
float moyenne;
printf ("donnez la valeur de la moyenne :");
scanf ("%f", &moyenne);
if (moyenne < 10)
printf (" mention ajourne \n");
else
      if (moyenne < 12)
      printf ("mention passable \n");
      else
            if (moyenne < 14)
            printf ("mention assez bien\n");
            else
                  if (moyenne <16)
                  printf ("mention bien\n");
                  else
                        printf ("mention tres bien\n");
return 0;
}
```

15. L'instruction de condition multiple : switch

La construction algorithmique appelée sélection ou choix multiple (SELON ou SUIVANT)

```
switch ExpressionEntiere
{
        case valeur1 : instruction1;
        break;
        case valeur2 : instruction2;
        break;
        ...
        default : instructionParDéfaut;
}
```

Notons que la clause **default** est optionnelle.

```
#include <stdio.h>

int main ()
{
char reponse[5];

printf ("1: saisie des données \n");
printf ("2: calcul sur les données \n");
printf ("3: afficher un récapitulatif \n");
printf ("q: quitter le programme \n");
scanf ("%s", reponse);
        switch (reponse[0])  // SELON la valeur de la première case du tableau
        {
        case '1' : printf ("les instructions de saisie du programme\n");
        break;
        case '2' : printf ("les  instructions de calcul du programme\n");
        break;
        case '3' : printf (" instructions pour récapitulatif du programme\n");
        break;
        case 'q' :
        case 'Q' : printf (" fin du programme\n");
        }

return 0 ;
}
```

16. Les boucles

a. La boucle While

TantQue (condition**)** instruction **FinTantQue**

Devient :

- Pour une seule instruction

while (condition)

instruction ;

```
#include <stdio.h>
int main ()
{
int note ;

printf ("donnez une valeur comprise entre 0 et 20 :");
scanf ("%d", &note);
        while ((note<0) || (note>20))
        scanf ("%d", &note);

printf ("Bravo, la valeur est acceptable");

return 0;
}
```

- Pour plusieurs instructions (un bloc d'instructions)

while (condition)

{

les instructions ;

}

```
#include <stdio.h>
int main ()
{
int note ;

printf ("donnez une valeur comprise entre 0 et 20 :");
scanf ("%d", &note);
        while ((note<0) || (note>20))
        {
        printf ("donnez une valeur comprise entre 0 et 20 :");
        scanf ("%d", &note);
        }

printf ("Bravo, la valeur est acceptable");

return 0;
}
```

b. La boucle do … While

Cette boucle n'est pas exactement celle que nous avons étudiée en algorithmique (**REPETER instructions… JUSQU'A (condition)**).

En C, Il s'agit de la boucle **FAIRE … Tant Que** (condition)

- Pour une seule instruction

do

instruction ;

while (condition) ;

/*Remarquez ce **;** pour marquer la fin de l'instruction */

```
#include <stdio.h>
int main ()
{
int note ;
printf ("donnez une valeur comprise entre 0 et 20 :");
scanf ("%d", &note);
        do
        scanf ("%d", &note);
        while ((note<0) || (note>20));

printf ("Bravo, la valeur est acceptable");
return 0;
}
```

- Pour plusieurs instructions

do
{
Les instructions ;
}
while (condition) ;
/*Remarquez ce **;** pour marquer la fin de l'instruction */

```
#include <stdio.h>
int main ()
{
int note ;
printf ("donnez une valeur comprise entre 0 et 20 :");
scanf ("%d", &note);
        do
        {
           printf ("donnez une valeur comprise entre 0 et 20 :");
           scanf ("%d", &note);
        }
        while ((note<0) || (note>20));

printf ("Bravo, la valeur est acceptable");
return 0;
}
```

c. La boucle FOR

- Pour une seule instruction

for (identificateur = ValeurInitiale; identificateur < ValeurFinale ; identificateur +=increment)

instruction ;

```
#include <stdio.h>
int main ()
{
int i, n;
printf ("Entrer un nombre pour entier positif:");
scanf("%d",&n);

printf ("\nles nombres pairs de 0 à %d sont :",n);
        for (i=0 ;  i <= n ;  i+=2)
        printf("%d ",i);

return 0;
}
```

- Pour plusieurs instructions

for (identificateur = ValeurInitiale; identificateur < ValeurFinale ; identificateur +=increment)

{

Les instructions ;

}

```
#include <stdio.h>
int main ()
{
int i, n;

printf ("Entrer un nombre pour entier positif:");
scanf("%d",&n);

printf ("\nles nombres de 0 à %d sont :",n);
   for (i=0 ; i <= n ; i++)
   {
      printf("le nombre numero %d : ", i);
      printf("%d \n", i);

   }
return 0;
}
```

Exercice 1 Ecrire un programme qui saisit deux entiers et affiche leur produit. Modifier ensuite le programme afin de saisir deux réels.

Exercice 2 Ecrire un programme qui échange deux entiers saisis. Afficher les entiers avant et après l'échange.

Exercice 3 Ecrire un programme qui affiche les code ASCII des lettres et des chiffres sous la forme suivante :

```
caractère = A     code = 65    code hexa = 41
caractère = B     code = 66    code hexa = 42
...
caractère = 1     code = 49    code hexa = 31
...
caractère = 9     code = 57    code hexa = 39
```

Exercice 4 Ecrire un programme qui détermine si un entier saisi est pair ou impair.

Exercice 5 Ecrire un programme qui affiche le plus grand de trois entiers saisis.

Exercice 6 Ecrire un programme qui affiche le plus grand et le plus petit d'une suite d'entiers saisis. Les nombres saisis ne se sont pas conservés en mémoire. La suite se termine avec la valeur 0.

Exercice 7 Ecrire un programme qui détermine tous les diviseurs d'un nombre entier saisi, plus grand que 1.

Exercice 8 Ecrire un programme qui simule l'opération de division entière entre deux entiers positifs a et b. Les deux nombres a et b sont saisis au clavier. On divise le plus grand par le plus petit, sans utiliser l'opérateur /. Afficher le quotient et le reste.

Exercice 9 Ecrire un programme qui multiplie deux entiers positifs a et b selon le principe récursif suivant :

```
a * b = a * (b-1) + a     si b est impair
a * b = (2 * a) * (b/2)   si b est pair et différent de 0

EXEMPLE : 36 * 7 = 36 * 6 + 36
                 = 72 * 3 + 36
                 = 72 * 2 + 108
                 = 144 * 1 + 108
                 = 144 * 0 + 252
                 = 252
```

Ecrire un programme qui lit deux entiers a et b à partir du clavier, et affiche leur produit selon l'algorithme itératif défini ci-dessus. Fournir les résultat tels qu'ils figurent dans l'exemple.

Exercice 10 Ecrire un programme se comportant comme une calculatrice, c'est-à-dire exécutant la boucle sur :

1. Lecture d'une ligne supposée contenant un entier, un opérateur et un entier (ex : 1 + 3). Les opérateurs sont +, -, *, \ et %.
2. Calcul de la valeur de l'expression.
3. Impression du résultat à l'écran.