1. Définition

Une instruction peut être :

- Soit une expression (pouvant comprendre une affectation, un appel de fonction...), terminé par un ; qui en fait signifie "on peut oublier le résultat de l'expression et passer à la suite",

- Soit une structure de contrôle (conditionnelle, boucle, branchement...),

- Soit un bloc d'instructions : ensemble de déclarations et instructions délimités par des accolades {}. Un bloc sera utilisé à chaque fois que l'on désire mettre plusieurs instructions là où on ne peut en mettre qu'une.

Seule la première forme est terminée par un ;. Un cas particulier est l'instruction vide, qui se compose uniquement d'un ; (utilisé là où une instruction est nécessaire d'après la syntaxe).

2.L'instruction if

La construction **if** est la construction logique de base du langage C qui permet d'exécuter un bloc d'instructions selon qu'une condition est vraie.

Exemple

```
if (i<0) {
c1=c1+10;
x1++;
}
```

Si la condition figurant entre parenthèses après le mot-clé **if** est vraie, alors le bloc d'instructions qui se trouve immédiatement après est exécuté, sinon aucune instruction n'est exécutée.

L'instruction d'un if peut être un autre if (imbriqué)

Exemple

```
if(c1)
{
i1;i2;
if (c2) i3;
}
```

3. L'instruction if-else

L'instruction **if-else** permet d'exécuter un bloc d'instructions selon qu'une condition est vraie ou fausse.
Le programme ci-dessous lit un nombre au clavier et indique si l'on a tapé un nombre négatif ou positif en refixant ce nombre à une valeur positive s'il était négatif:

```
#include <stdio.h>
void main ()
{
int i;
printf("Tapez un nombre entier positif ou negatif: ");
scanf("%d", &i);
if (i<0) {
i=-i;
printf("J'ai remis i à une valeur positive.\n");
} else {
printf("Vous avez tapé un nombre positif.\n");
}
}
```

Si la condition figurant entre parenthèses après le mot-clé **if** est vraie, alors le bloc d'instructions qui se trouve immédiatement après est exécuté, sinon c'est le second bloc qui se trouve après le **else** qui est exécuté.

Exemple

```
if(c1) i1;
else if (c2) i2;
else if (c3) i3;
else i4;
i5;
```

Ce qui signifie : si c1 alors i1 puis i5, sinon mais si c2 alors i2 puis i5, ... Si ni c1 ni c2 ni c3 alors i4 puis i5.

Remarque

On n'utilise pas de point-virgule après l'accolade fermante d'un bloc d'instructions. Si un bloc d'instructions se réduit à une seule instruction alors on peut omettre les accolades de délimitation:

```
#include <stdio.h>
void main ()
{
int i;
printf("Tapez un nombre entier positif ou negatif: ");
scanf("%d", &i);
if (i>=0)
printf("Vous avez tapé un nombre positif.\n");
```

```
else
printf("Vous avez tapé un nombre negatif.\n");
}
```

Le tableau suivant rassemble les divers opérateurs logiques opérant sur des nombres et des variables numériques:

a < b	Vrai si a strictement inférieur à b
a > b	Vrai si a strictement supérieur à b
a <= b	Vrai si a inférieur ou égal à b
a >= b	Vrai si a supérieur ou égal à b
a == b	Vrai si a strictement égal à b
a != b	Vrai si a différent de b

Opérateurs logiques numériques

Des propositions logiques telles que a<12 ou i>=j peuvent être combinées entre elles au moyen de connecteurs logiques **&&** (*et*), || (*ou*) et **!** (*négation*) pour former des expressions logiques complexes, appelées aussi *expressions booléennes*. De telles expressions sont évaluées de gauche à droite dans l'ordre naturel de lecture,
l'utilisation de parenthèses permet de mieux contrôler l'ordre d'évaluation des expressions, il ne faut donc pas hésiter à les utiliser en cas de doute sur la priorité des opérateurs employés.

Par exemple les formules suivantes:
`!(i <= 0) || (i >= 10)`
et
`!((i <= 0) || (i >= 10))`
ne sont pas équivalentes Mais la dernière est équivalente à `(i > 0) && (i < 10)`

Exercices

Exo1: Faire un programme qui demande deux nombres au clavier et qui affiche 'divisible' si le premier est divisible par le deuxième. Conseil: pensez à l'opérateur %.

```
#include <stdio.h>
void main ()
{
int m,n;
/* lecture des données */
printf("donnez deux nombres entiers positifs: \n");
scanf("%d %d",&m,&n);
/* test de divisibilité et affichage */
if (m%n==0) {
printf("divisible \n");
else
printf("non divisible \n");
}
}
```

Exo2: Faire un programme qui lit deux nombres et qui teste si ces nombres sont compris dans l'intervalle [-5, +5]. Sinon on affecte le premier à -5 si la première valeur donnée est plus petite que -5 et le deuxième à +5 si la deuxième valeur est plus grande que 5. Imprimer ensuite la liste des nombres, du premier au deuxième nombre.

Exo3: Ecrire un programme qui résout les équations du second degré (ax2 + bx + c = 0). Le programme demande a, b et c à l'utilisateur puis indique le nombre de solutions ainsi que leurs valeurs.

Remarque

Variable booléenne. Toute expression retournant une valeur logique ou entière est une condition valable dans une construction if. C'est pourquoi il est tout à fait légal et même fréquent de stocker le résultat d'une condition dans une variable de type int et d'utiliser plus loin cette variable comme condition d'un if:

```
#include <stdio.h>
void main ()
{
float x;
int plusgrand;
printf(‘‘Entrez un reel: ’’);
scanf(‘‘%f’’,&x);
plusgrand = (x>15.0);
if (plusgrand)
printf (‘‘Plus grand\n’’);
else
printf (‘‘Plus petit\n’’);
}
```

Le langage C considère toute valeur numérique entière non nulle comme étant une valeur logique vraie, seule la valeur numérique 0 est considérée comme étant une valeur logique fausse, de ce fait l'expression plusgrand = (x>15.0) dépose dans la variable plusgrand une valeur non nulle si x est plus grand que 15.0 et la valeur 0 sinon. De plus, pour les mêmes raisons, une expression telle que if (plusgrand) n'est en fait qu'un raccourci pour if (plusgrand != 0)

4. Instruction switch

L'instruction **switch** est l'instruction de contrôle la plus souple du langage C. Elle permet d'exécuter différentes instructions en fonction d'une expression qui pourra avoir plus de deux valeurs. Une instruction de contrôle comme if ne peut évaluer que deux valeurs d'une expression: vrai ou faux. Dans le cas où l'on souhaite exécuter une action différente selon les différentes valeurs possibles d'une variable cela oblige à utiliser une cascade de **if**...**else** comme l'illustre l'exemple suivant:

```
#include <stdio.h>
void main ()
{
char operation;
int r, a, b;
printf("Entrez un signe d'opération: ");
scanf("%c", &operation);
a=273; b=158; r=0;
if (operation=='+' || operation=='p')
r = a + b;
else if (operation=='-' || operation=='m')
r = a - b;
else if (operation=='*' || operation=='f')
r = a * b;
else if (operation=='/' || operation=='d')
r = a / b;
else
printf("Non valable: ");
printf("%d %c %d = %d\n", a, operation, b, r);
}
```

L'instruction switch permet de résoudre ce problème de façon plus générale:

```
#include <stdio.h>
void main ()
{
char operation;
int r, a, b;
printf("Entrez un signe d'opération: ");
scanf("%c", &operation);
a=273; b=158; r=0;
switch (operation) {
case '+':
case 'p':
r = a + b;
break;
case '-':
case 'm':
r = a - b;
break;
case '*':
case 'f':
r = a * b;
break;
case '/':
case 'd':
r = a / b;
break;
default:
printf("Non valable: ");
}
```

```
printf("%d %c %d = %d\n", a, operation, b, r);
}
```

Cette instruction fonctionne en examinant le contenu de la variable située après le mot-clé **switch**, dans le cas présent operation. Ce contenu est successivement comparé aux valeurs figurant après chacune des clauses case. Si une de ces clauses comporte une valeur identique alors les instructions figurant après cette clause sont exécutées y compris celles figurant après les clauses case suivantes. L'instruction break (*interrompre*) provoque une sortie immédiate du **switch**. de façon à ce que l'exécution se poursuive après l'accolade fermante du **switch**. **break** termine généralement chacun des blocs d'instructions correspondant à chacun des cas prévus car, sauf exception, on ne souhaite exécuter que les instructions figurant immédiatement après un cas donné.
Si operation ne correspond à aucun des cas prévus, ce sont les instructions figurant après le mot clé **default** qui sont exécutées, dans le cas présent l'affichage d'un message d'erreur au moyen de la fonction **printf.**
Les clauses **default** et **break** sont optionnelles.

Exercice

Réecrire ce programme en utilisant l'instruction switch

```
#include <stdio.h>
 int
 main (int argc, char *argv[])
 {
 int i;
 scanf ("%d", &i);
 if (i == 6 || i == 8)
 {
 printf ("le nombre est superieur a 5\n");
 printf ("le nombre est pair\n");
 }
 else if (i == 4 || i == 2 || i == 0)
 printf ("le nombre est pair\n");
 else if (i == 7 || i == 9)
 {
 printf ("le nombre est superieur a 5\n");
 printf ("le nombre est impair\n");
 }
 else if (i == 5 || i == 3 || i == 1)
 printf ("le nombre est impair\n");
 else
 printf ("ceci n est pas un nombre\n");
 return 0;
 }
```

Solution

```
#include <stdio.h>
int
 main (int argc, char *argv[])
 {
 int i;
 scanf ("%d", &i);
 switch (i)
 {
 case 6:
 case 8:
 printf ("le nombre est superieur a 5\n");
 case 0:
 case 2:
 case 4:
 printf ("le nombre est pair\n");
 break;
 default:
 printf ("ceci n est pas un nombre\n");
 break;
 case 9:
 case 7:
 printf ("le nombre est superieur a 5\n");
 case 5:
 case 1:
 case 3:
 printf ("le nombre est impair\n");
 break;
 }
 return 0;
 }
```

5. Les instructions de boucle

Ces instructions permettent de répéter un bloc d'instructions entre accolades, appelé *corps de boucle*, un certain nombre de fois, tant qu'une condition est vérifiée. Il existe plusieurs instructions de boucle.

5.1 Instruction *while*

La plus simple est l'instruction **while** (*tant que*). Elle se présente sous la forme suivante:

```
while (condition) {
...
}
```

Exemple 1

```
long nombreEntre = 0;
while (nombreEntre != 47)
{   printf("Tapez le nombre 47 ! ");
```

```
    scanf("%ld", &nombreEntre);
}
```

Exemple 2

```
long compteur = 0;
while (compteur < 10)
{
    printf("Salam !\n");
    compteur++;
}
```

Remarque

La boucle **while** pourrait très bien ne jamais être exécutée si la condition est fausse dès le départ. Par exemple, si on avait initialisé le compteur à 50, la condition aurait été fausse dès le début et on ne serait jamais rentré dans la boucle.

5.2 Instruction do- *while*

Une autre instruction similaire se présentant sous la forme suivante:

```
do-while (faire-tant que)
do {
...
} while (condition);
```

Exemple

```
long compteur = 0;

do
{
    printf("Salam !\n");
    compteur++;
} while (compteur < 10);
```

Remarque

Pour la boucle **do... while**, c'est différent : Cette boucle s'exécutera toujours au moins une fois. En effet, le test se fait à la fin comme vous pouvez le voir. Si on initialise compteur à 50, la boucle s'exécutera une fois.

Les instructions **while** et **do-while** présentent une différence subtile. Dans le cas de l'instruction **while**, la condition est tout d'abord examinée, si elle est vraie alors le bloc d'instructions est exécuté, après quoi la condition est de nouveau évaluée et le bloc d'instructions est de nouveau exécuté et ainsi de suite tant que la condition est vraie. Dans le cas de l'instruction **do-while**, par contre, le bloc d'instructions est d'abord exécuté **puis** la condition est évaluée. Si elle est vraie, alors le bloc d'instructions est de nouveau exécuté puis la condition de nouveau examinée et ainsi de suite. On s'aperçoit ainsi que si la condition est fausse lors de sa première évaluation alors le bloc d'instructions n'est

jamais exécuté dans le cas de l'instruction **while** alors qu'il l'est au moins une fois dans le cas de l'instruction **do-while**.

Exemple

```
#include <stdio.h>
void main ()
{
int i, M, N;
do {
printf("Donnez M N: ");
scanf("%d %d", &M, &N);
} while (M <= -6 || N >= 6 || M >= N);

i = M;
while (i <= N) {
printf("%d ", i);
i = i+1;
}
printf("\n");
}
```

Les instructions placées dans le bloc entre **do-while** demandent à l'utilisateur de rentrer deux nombres puis on teste les conditions d'exclusion (M plus petit ou égal à 6 ou bien N plus grand ou égal à 6 ou bien M plus grand ou égal N). Si une de ces conditions est vérifiée, on redemande les nombres à nouveau. Si ces conditions d'exclusion ne sont pas vérifiées, alors on continue plus loin.

La boucle **while** simple suivante a un fonctionnement similaire, à la nuance près évoquée plus haut. Avant toute chose, la condition figurant entre parenthèses est évaluée. Si elle est vraie, c'est-à-dire si la valeur courante de la variable i est inférieure à la valeur courante de la variable N, alors les instructions se trouvant entre accolades sont exécutées: la valeur de i est affichée et la valeur courante de i est augmentée de 1. La condition est alors de nouveau évaluée. Si elle est toujours vraie on exécute à nouveau les instructions du bloc. Le programme se poursuit ainsi jusqu'à ce que la condition ne soit plus vérifiée ce qui ne manquera pas d'arriver car la valeur de i est augmentée d'une unité chaque fois que le bloc d'instructions entre accolades est exécuté alors que la valeur de N, elle, n'est pas modifiée.

5.3 Instruction for

Le langage C offre pour cela une instruction pratique: l'instruction **for**. Dans son utilisation la plus courante, elle se présente comme suit:

```
for (i= m1; i<= m2; i++) {
...
}
```

Cette construction permet de répéter le bloc d'instructions entre accolades, appelée *corps de boucle*, un certain nombre de fois, la variable i, appelée *compteur de boucle*, prenant

une valeur différente à chaque tour de boucle: m1 au premier tour, m1+1 au deuxième, m1+2 au troisième et ainsi de suite jusqu'à m2 compris. Le corps de boucle est ainsi exécuté (m2-m1+1) fois. La construction i++ incrémente la valeur de la variable i de une unité.

Exemple 1

```
#include <stdio.h>
#include <math.h>
#define pi 3.14156
void main ()
{
int i;
float x;
for (i=1; i<=10; i++) {
x = sin(i*pi/180.0);
printf("sin(%2d) = %f\n", i, x);
}
}
```

Le programme ci-dessus tabule les valeurs des sinus des 10 premiers degrés d'angle.

Exemple 2
La boucle suivante affiche les 5 premières puissances de 2:

```
for (i=1; i<=32; i=i*2) {
printf("%d ", i);
}
```

Remarque
il est possible de réaliser au moyen de la boucle **for** toutes sortes de compteurs, par exemple la construction suivante est une boucle décroissante exécutée 5 fois, le compteur, i, variant de 5 à 1:

```
for (i=5; i>0; i--) {
...
}
```

Exercice 1
Ecrire un programme retournant la factorielle d'un nombre entier entré par l'utilisateur.
Solution

```
#include<stdio.h>
int main(void)
{
  int i,n,res;
  res=1;
  printf("Entrer un entier :\n");
  scanf("%d",&n);
```

```
  for(i=n;i>1;i--)
  {
     res=res*i;
  }
  printf("%d ! vaut %d\n",n,res);
  return 0;
}
```

Exercice 2

Ecrire un programme retournant le PGCD ainsi que le PPCM de 2 entiers entrés par l'utilisateur.

Solution

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

int main(void)
{
  int a0,a,b0,b,r;

  printf("Entrer les coefficients a et b :");
  scanf("%d %d",&a0,&b0);
  a=a0;
  b=b0;
  while (b>0)
  {
     r=a%b;
     a=b;
     b=r;
  }
  printf("\nLe pgcd de %d et de %d vaut %d",a0,b0,a);
  printf("\nLe ppcm de %d et de %d vaut %d\n",a0,b0,(a0*b0)/a);
  return 0;
}
```