DI GALLO Frédéric

Programmation Classique en langage C

Cours du Cycle d'Approfondissement

CNAM ANGOULEME 2000-2001

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PROGRAMMATION CLASSIQUE :

LANGAGE C

PROGRAMMATION CLASSIQUE EN C

INFORMATIQUE – CNAM ANGOULEME 2000-2001

PROGRAMMATION CLASSIQUE EN C

I. LES VARIABLES DE TYPE SIMPLE

On rappelle qu'en programmation classique, pour mémoriser une valeur en mémoire centrale, il est nécessaire d'introduire une variable. Celle-ci est définie au moyen de deux éléments :

- son nom – encore appelé identificateur - qui permet d'accéder à la valeur qu'elle contient et de la manipuler dans l'algorithme ou dans le programme,
- son type qui précise la nature de la valeur qu'elle peut accueillir.

Par exemple en C, la déclaration suivante définit deux variables destinées à accueillir respectivement un entier et un caractère. `int I ; char C ;`

Remarque : le langage C distingue les lettres minuscules des lettres majuscules. Les identificateurs i et I désignent donc deux variables différentes.

Le type d'une variable induit plus précisément la représentation de la valeur en mémoire centrale et donc l'espace - en terme d'octets - monopolisé par cette valeur. *Ainsi, le type « int » permet d'accueillir un entier codé en arithmétique signée sur deux octets. Le type « char » utilise quant à lui le code ASCII (un octet) pour représenter un caractère*.

La déclaration d'une variable peut donc être assimilée à la réservation en mémoire centrale d'un emplacement dont la taille dépend de son type. *On peut représenter de la manière suivante les conséquences de la déclaration des deux variables I et C.*

Pour la mémorisation des nombres, le langage C propose différents types dont la taille dépend de l'implémentation en machine. Le tableau ci-dessous présente les caractéristiques des types numériques proposés par TURBO C.

Nom du type	Nature de la valeur	Nbre d'octets	Domaine
int	Entier signé	2	-32 768 ; 32 767
short	Entier signé	2	-32 768 ; 32 767
long	Entier signé	4	-2 147 483 648 ; 2 147 483 647
float	Réel	4	$3.4\ 10^{-38}$; $3.4\ 10^{38}$
double	Réel	8	$1.7\ 10^{-308}$; $1.7\ 10^{308}$

Il est important de noter qu'en C la déclaration d'une variable n'entraîne pas son initialisation et le compilateur signalera une erreur si le programmeur la référence sans l'avoir initialisée. L'initialisation d'une variable peut se faire au moment de sa déclaration de la manière suivante :

NomDuType NomVariable=Valeur;

Les valeurs entières sont de type « int », mais on peut leur affecter un type « long » en suffixant leur nom d'un « l » ou d'un « L ». *Par exemple la valeur « 32L » désigne une valeur entière de type « long »*. De même les valeurs réelles sont par défaut de type « double », mais on peut leur affecter un type « float » en suffixant leur nom d'un « f » ou d'un « F ». Lors d'une initialisation, il est important de veiller à la compatibilité du type de la valeur et de la variable. Le langage C réalise des conversions de type avec éventuellement perte d'information.

Par exemple la déclaration « int I=78 934 » est acceptée par le compilateur C mais entraîne l'affectation de la valeur 13 398 dans la variable I. La déclaration « float tva=5.5 » est aussi acceptée alors que la valeur 5.5 est de type « double » alors que la variable destinée à accueillir la variable est de type « float ». Il serait plus juste d'écrire « float tva=5.5f ».

Application 1: On considère le programme C suivant :

```
void main()
{
    int I=32767 ;
    long J= 48L ;
    char C='A',NL='\n' ;
    }
```

Remarque : la constante '\n' désigne le caractère non visualisable 'LF'. Son affichage provoque un changement de ligne.

a) Indiquer pour chaque variable la représentation en binaire de la valeur qu'elle contient.

I: 7F FFh
J: 00 00 00 30h
C: 41h
NL: 0Ah

b) On enrichit le programme de l'instruction qui consiste à incrémenter le contenu de la variable I. Donner en binaire et en décimal la nouvelle valeur contenue dans I.

I: 7F FF + 1 = -32768 (ça revient en arrière puisque le C boucle).

c) Expliquer, pourquoi le langage C permet de comparer deux caractères et autorise de telles écritures : C='A'+1 ;

Pour le langage C, 'A' correspond au code 65h, il fait 65+1 = 66h or 'B'=66h.
Si l'on teste C<'E', la réponse sera oui car 'A'=65h est inférieur à 'E'=69h.

Application 2: Codage du sexe d'une personne.

Pour coder le sexe d'une personne, certains organismes utilisent les valeurs 1 et 2. Proposer, en C, la déclaration d'une variable destinée à mémoriser ce code.

On choisit un caractère pour limiter la taille à un octet.

II. LES VARIABLES DE TYPE STRUCTURE

Une variable de type structuré permet de mémoriser plusieurs valeurs, qui ont un lien entre elles, en les regroupant sous un nom unique. Par exemple pour mémoriser les N notes d'un élève il est plus judicieux d'utiliser la variable de type structuré « Notes » qui regroupe toutes les notes, plutôt que d'utiliser N variables de type simple.

Les deux grands types structurés sont :

- **le tableau** (à 1 ou à 2 dimensions) qui regroupe des valeurs de même type,
- **l'enregistrement** qui est composé de valeurs ayant des types différents.

Le tableau d'enregistrements est un dérivé des deux types précédents puisqu'il s'agit d'un tableau à 1 dimension dans lequel chaque composant est un enregistrement.

Comme pour les variables de type simple, toute déclaration d'une variable de type structuré entraîne la réservation en mémoire centrale d'un emplacement dont la structure et la taille dépend du type. Le tableau ci-dessous présente des exemples de variables de types structurés.

Tableau à 1 dimension T de 5 entiers	Tableau à 2 dimensions TD d'entiers (4 lignes et 3 colonnes)	Enregistrement H composé de deux champs : heure, minute
0 1 2 3 4	0 1 2 / 0 1 2 3	heure | minute
int T[5];	int TD[4][3];	struct horaire { int heure; int minute; }; struct horaire H;

L'accès aux valeurs contenues dans une variable de type structuré est réalisé en mentionnant le nom de la variable suivi d'une information qui identifie la valeur dans la variable. *Par exemple, pour accéder au $i^{ème}$ composant du tableau T à 1 dimension on utilise la référence T[i+1] car le premier composant porte le numéro 0. Pour accéder au composant « heure » de l'enregistrement H, il faut utiliser la notation : H.heure.*

Exemple de tableau d'enregistrement: Famille

	Nom	Prénom	Age
0	C	B	18
1	B	O	17
2	C	B	23

Struct personne Famille[30]

Famille[2].age *Famille[0]*

Remarque : pour mémoriser une chaîne de caractères l'une des possibilités en C est de déclarer un tableau à 1 dimension de caractères. *Par exemple la déclaration « char Mess[25] » permet de stocker un message d'au plus 25 caractères dans la variable Mess.*

Application 1: On considère le programme C suivant :

```
void main()
{       struct LigneCommande
        {       int NumeroProduit ;
                int NumeroCommande ;
                int Qte ;
        };
        struct LigneCommande T[100] ;
}
```

Calculer en octets l'espace monopolisé par le tableau T.

LigneCommande = 6 octets car NumeroProduit, NumeroCommande, Qte sont des entiers (2 octets). Ce qui donne pour T[100]: 600 octets.

Application 2: On souhaite utiliser le tableau Decomp:

à 1 dimension pour mémoriser la décomposition binaire d'un entier (type : « int ») saisi au clavier. Par exemple la décomposition de la valeur 23 sera mémorisée ainsi :

0	1
1	0
2	1
3	1
4	1
5	
6	
7	

Remarque : le bit de plus fort poids est placé dans le composant 0 du tableau Decomp.

Combien de composants devra-t-on déclarer pour le tableau Decomp afin qu'il soit en mesure de mémoriser la décomposition binaire de tout entier saisi au clavier ?

Un entier correspond à 2 octets donc 16 bits. Decomp devra donc être en mesure de contenir 16 composants. On le déclare par int Decomp[16].

III. INTERET des TYPES SIMPLES & STRUCTURES

Dans de nombreuses informatisations, l'une des étapes fondamentales du développement consiste à passer d'une représentation externe de l'information à une représentation interne en mémoire centrale ou sur support à stockage permanent. Cette représentation interne doit permettre d'une part de retrouver la représentation externe de l'information, moyennant l'application d'un processus déterminé, et d'autre part de mettre en œuvre, le plus facilement possible, tous les traitements que l'on souhaite réaliser sur l'information. Elle doit en outre optimiser l'espace mémoire.

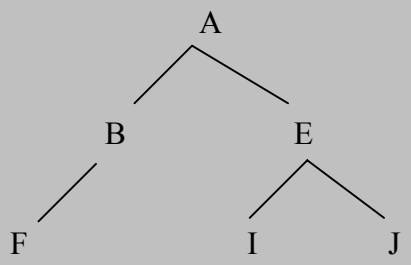
Les représentations internes en mémoire centrale sont basées sur tous les types simples ou structurés vus précédemment. *Ainsi pour implémenter en mémoire centrale le polynôme : $5x^4 + 3x + 2$, on pourra envisager les deux solutions illustrées et commentées ci-dessous :*

<table>
<tr>
<td>On introduit la variable P de type chaîne de caractères suivante :
5x4 +3x+2</td>
<td>On introduit le tableau d'enregistrements P suivant :
<table>
<tr><th>Coeff</th><th>Degré</th></tr>
<tr><td>5</td><td>4</td></tr>
<tr><td>3</td><td>1</td></tr>
<tr><td>2</td><td>0</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>Cette implantation ne se prête pas à une exploitation facile car l'extraction des monômes constituant le polynôme impose la mise en œuvre d'un traitement complexe sur les chaînes de caractères.</td>
<td>Cette représentation interne est satisfaisante et permet de développer sans trop de difficultés toutes les opérations sur un polynôme.</td>
</tr>
</table>

Application 1: Proposer une représentation interne :

en mémoire centrale pour implanter un arbre binaire. Un arbre binaire est un arbre dont chaque sommet admet au plus deux fils.

Exemple d'arbre binaire :



La représentation proposée devra permettre une exploitation simple de l'arborescence et dans un même temps une optimisation de la mémoire. L'exploitation concerne les opérations élémentaires que l'on peut effectuer sur un arbre c'est-à-dire :

- la consultation du père, des fils et des frères d'un sommet,
- l'ajout et la suppression d'un sommet.

Avec racine: 0

Info	Fils gauche	Fils droit
A	1	2
B	3	0
E	4	5
F	0	0
I	0	0
J	0	0

Application 2: Proposer une représentation interne :

pour implanter un jeu de 32 cartes. Cette représentation devra permettre de mettre en œuvre n'importe quel jeu de cartes avec ses propres règles.

Une carte est représentée par sa valeur et sa couleur:

	valeur	couleur
0	1	Carreau
1	7	Pique
2	dame	Trèfle
…	...	…
31	.	.

IV. LES INSTRUCTIONS DU LANGAGE

Les instructions du langage permettent de développer des traitements sur des variables de tous les types vus précédemment. En C chaque instruction se termine par un point virgule.

Ces instructions peuvent être classées en 4 grandes catégories :
- l'instruction d'affectation,
- les instructions d'entrée-sortie,
- l'alternative,
- l'itérative

4.1) L'instruction d'affectation

Elle permet d'affecter dans une variable de type simple ou dans un composant d'une variable de type structuré le résultat d'une expression. Le type de ce résultat doit être compatible avec le type de la variable.
Syntaxe : Variable=Expression;

L'expression peut porter sur des nombres ou des chaînes de caractères. Dans le cas d'une expression de type arithmétique, celle-ci peut faire intervenir 3 types de composants :
- des références à des variables numériques de type simple ou à des composants de variables de type structuré,
- des constantes, encore appelées littéraux,
- des opérateurs.

Les opérateurs arithmétiques:

Le tableau ci-dessous récapitule les opérateurs arithmétiques autorisés en C :

Symbole	Opération
+	addition
-	soustraction
*	multiplication
/	division
%	Modulo (reste de la division euclidienne)
++	incrémentation
--	décrémentation

Les opérateurs +, -, *, / peuvent être combinés avec l'affectation. Ainsi l'expression « i += j » est équivalente à « i = i + j ». Les deux derniers opérateurs sont unaires. Leur rôle est d'incrémenter ou de décrémenter le contenu de la variable. Ainsi, les expressions ++i ou i++ ajoute une unité à la variable i. Lorsqu'ils sont intégrés à l'expression d'une affectation il est important de noter qu'ils modifient le contenu de la variable à laquelle ils sont attachés. Dans une telle situation l'affectation ne modifie plus seulement le contenu de la variable placée à gauche du signe =. Selon la position de l'opérateur par rapport à la variable (avant ou après) l'évaluation de l'expression composant le membre droit de l'affectation prendra la valeur de la variable avant qu'elle soit modifiée ou après.

Dans l'exemple suivant les deux affectations modifient le contenu de la variable i qui prend la valeur 3. La première affectation renvoie la valeur 2 à j alors que la seconde renvoie la valeur 3.

```
void main()
{   int      i =2;
    int      j ;
    j=i++ ;
    j=++i ;
}
```

En plus des opérateurs cités dans le tableau ci-dessus, il existe l'opérateur ternaire qui permet l'inclusion d'une expression conditionnelle dans le calcul d'une expression. Sa syntaxe a la valeur suivante : **(expression logique) ? ExpressionSiVrai : ExpressionSiFaux**.

Par exemple, l'expression (i>0) ?i*2 :i+2, renvoie le produit de i par 2 si i est strictement positif et le résultat de l'ajout de 2 à i si i est négatif ou nul.

Les opérateurs de comparaison:

Dans l'expression logique, on peut mentionner les opérateurs de comparaison classiques : <, >, <=, >=, ==(égal), != (différent) ainsi que les opérateurs booléens : &&(et) et ||(ou).

Application 1: Remise de 20% pour plus de 3 articles achetés :

La société X applique une remise de 20% si le nombre d'articles achetés est strictement supérieur à 3. Ecrire l'expression qui permet de valoriser la variable MontantAPayer à partir des deux variables supposées renseignées : TotalAchats et NbProduits qui représentent respectivement le montant total des achats et le nombre d'articles achetés.

```
void main()
{  int NbProduits;
   float MontantAPayer, TotalAchat;

   Saisie de NbProduits et TotalAchat

   MontantAPayer = (NbProduits>3)?TotalAchats-(TotalAchats*0.20):TotalAchats;
}
```

Auto-évaluation 1:

L'évaluation de toute expression repose d'une part sur les règles de priorité entre les opérateurs en l'absence d'un parenthèsage et d'autre part sur les types des opérandes qui constituent l'expression. Pour ce dernier point, il est utile de savoir que l'ordinateur ne sait calculer une expression arithmétique de la forme « opérande1 opérateur opérande2 » que lorsque les deux opérandes sont du même type. Lorsque ce n'est pas le cas, l'ordinateur transforme le type de l'un des opérandes pour que les deux membres deviennent du même type. La conversion appliquée suit une hiérarchie destinée à ne pas perdre l'information. Ainsi lorsque l'on a une variable de type « float » et une variable de type « int » il y aura conversion automatique de la seconde variable en « float ». Le résultat obtenu est toujours du même type que celui des deux opérandes. Avec un tel principe le résultat de l'expression « 5/2 » est égal à 2 et non à 2.5 car 5 et 2 sont des valeurs de type « int ». En revanche, l'expression « 5/2f » fournit 2.5 comme résultat car l'ordinateur convertit en float les deux opérandes.

Application 1: On considère le programme C suivant :

```
void main()
{    int i=5,j=2,k ;
     float w,x,y,z ;
     w=i/2+1;
     x=i/2f+1;
     y=++i/2 +1/2f;
     z=i++/j + 1/2;
}
```

Fournir le contenu des variables w, x, y et z à l'issue des différentes affectations executées.

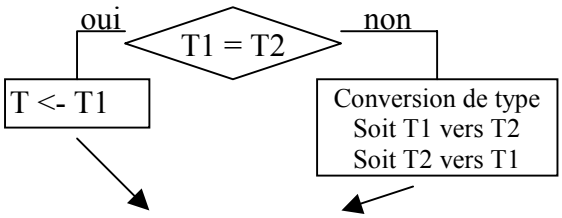
w=3; x=3.5; y=3+0.5=3.5; z=3; i=7;

L'opérateur de transtypage:

L'opérateur de cast – encore appelé opérateur de transtypage - permet de modifier le type d'une variable dans une expression arithmétique. Cet opérateur qui a une priorité supérieure aux opérateurs arithmétiques usuels s'écrit en entourant par des parenthèses le type vers lequel on veut transformer une expression. Le programme ci-dessous illustre le principe de cet opérateur.

```
void main()
{   int i=5,j=2
    float x ;
    x=i/j;                    /* le résultat est 2 car i et j sont des entiers ; le résultat est donc un entier */
    x=(float)i/j;        /* le résultat est 2.5 car l'opérateur (float) (prioritaire par rapport à la division) */
                              /* a transformé le  type de la variable i en float ; le résultat est donc un float */
    x=(float)(i/j);      /* le résultat est 2 car la division est évaluée et donne un entier qui est ensuite */
                                                                                    /* transformé en réel */
}
```

I + J
Opérande 1 – Opérateur – Opérande 2
Type T1 Type T2



Le résultat est de type T

Sens de la conversion int ---------> float ------>double

Application 1: On considère les expressions arithmétiques suivantes :

```
1.5+(float)i/j;
1.5+(float)(i+1)/(float)j +5/i;
1+(float)(i/2f)+(float)(i/j);
```

a) En supposant que i et j sont deux variables entières qui contiennent respectivement les valeurs 5 et 2, fournir le résultat retourné par chaque expression.

1.5+(float)i/j;	4
1.5+(float)(i+1)/(float)j +5/i;	1.5+3+1= 5.5
1+(float)(i/2f)+(float)(i/j);	1+2.5+2= 5.5

b) Indiquer les cast qui sont inutiles.

```
1.5+(float)i/j;
1.5+(float)(i+1)/(float)j +5/i;
1+(float)(i/2f)+(float)(i/j);
```

4.2) Les instructions d'entrée-sortie

L'instruction **scanf()**:

Elle permet d'acquérir en mémoire centrale une valeur. Cette valeur sera placée dans une variable de type simple ou dans un composant d'une variable de type structuré. Le format précise le type de valeur qui doit être saisi et affecté dans la variable. Le tableau ci-dessous présente les formats associés aux différents types étudiés précédemment.

Syntaxe : scanf(« format »,&Variable)

Type	Format
int ou short	%d
long	%D
float ou double	%f
char	%c

Par exemple, l'instruction *scanf (« %d »,&I)* permet d'acquérir un entier et de le stocker dans la variable I de type « int ». L'expression « &Variable » correspond à l'adresse de la variable. C'est le contenu de cette adresse qui sera valorisée par la fonction scanf ().

L'instruction **printf()**:

Elle permet d'afficher à l'écran un message pouvant intégrer des expressions faisant référence à des variables de type simple ou à des composants de variables de type structuré. Le format précise le type de la valeur contenue dans la variable citée. Il peut prendre de nombreuses valeurs mais on se limitera aux mêmes valeurs que pour l'instruction scanf().

Syntaxe : printf (« Message format », variable ou expression)

Par exemple, l'instruction *printf (« le double de %d est %d\n», I, I*2)* affiche le message « le double de 5 est 10 » si le contenu de la variable I est 5. Le caractère « \n » provoque un retour à la ligne après l'affichage du message.

Application 1: Ecrire le programme C :

qui calcule et affiche la surface ainsi que le périmètre d'un rectangle dont on saisit la longueur et la largeur. On suppose que ces valeurs sont des entiers.

```
#include <stdio.h>

void main()
{
  float surface, perimetre, longueur, largeur;
  clrscr();
  printf ("Saisir la longueur: ");
  scanf ("%f",&longueur);
  printf ("\nSaisir la largeur: ");
  scanf ("%f",&largeur);
  perimetre=2*(longueur+largeur);
  surface=longueur*largeur;
  printf("\n\nle p,rimŠtre est %f et la surface est %f",perimetre,surface);
  getch();
}
```

Application 2: Ecrire le programme C :

qui calcule la note arrondie au ½ point supérieur d'une note saisie au clavier.

```
#include <stdio.h>

void main()
{
  float note,notea,i;
  clrscr();
  printf ("Saisir la note: ");
  scanf ("%f",&note);
  i=note-(int)note;
  notea=(i==0)?note:(i>0.5)?(int)note+1:(int)note+0.5;
  printf ("\nLa note arrondie est %f",notea);
  getch();
}
```

4.3) Les instructions de choix

L'instruction conditionnelle IF:

La syntaxe générale de l'alternative a la forme suivante :

```
if (expression booléenne)
  {
        bloc1 d'instructions   /* si le résultat de l'expression booléenne est VRAI les instructions */
  }                            /*du bloc1 sont exécutées, sinon les instructions du bloc 2 sont exécutées*/
else
  {
        bloc2 d'instructions
  }
```

Remarques :

- Chaque instruction à l'intérieur d'un bloc se termine par un point virgule.
- La partie « else » est facultative.
- Dans le cas où le bloc est formé d'une seule instruction il n'est pas utile de mentionner les accolades.
- Les principaux opérateurs relationnels que l'on peut mentionner dans l'expression booléenne ont été cités lors de l'étude de l'opérateur ternaire.

L'instruction de sélection SWITCH:

Lorsque l'on doit tester le contenu d'une variable avec de nombreuses valeurs et déclencher des traitements particuliers pour chacune de ces valeurs, il est recommandé d'utiliser l'instruction « **switch** » plutôt qu'une imbrication d'instructions « **if …else …** ». La syntaxe de cette instruction a la forme suivante :

```
switch (NomVariable)
{case Valeur1 : {bloc d'instructions}
  case Valeur2 : {bloc d'instructions}
   …
  default : {bloc d'instructions}
}
```

L'instruction **switch**, compare la valeur contenue dans la variable avec chacune des valeurs citées après le mot clé « **case** ». Dès qu'elle trouve la correspondance elle exécute le bloc d'instructions associé. La dernière instruction associée à un bloc doit être « **break** » afin de passer à l'instruction qui suit le switch. Dans le cas où aucune correspondance n'est trouvée, l'instruction « switch » exécute soit le bloc d'instructions associée à la clause « **défault** » si celle-ci est mentionnée, soit passe à l'instruction qui suit l'instruction « switch ».

Application 1: Ecrire le programme de l'exemple d'exécution suivant :

(les valeurs saisies sont soulignées) :

Opérande 1 : 124
Opérateur (+, *, -): +
Opérande2 : 35
Résultat : 159

On suppose que les opérandes saisis sont des entiers de type « int ».

```
#include <stdio.h>      /* Appel à la bibliothèque stdio */

void main()
{
 int a,b,result,B=0,encore=0;
 char operateur;

 do {
 clrscr();                          /* efface l'écran */
 printf ("\nop,rande nø1 : ");      /* Affiche l'interrogation */
 scanf ("%d",&a);                   /* récupère l'opérande date saisie par l'utilisateur */
 printf ("\nop,rateur (+,*,-) : ");
 scanf ("%s",&operateur);
 printf ("\nop,rande nø2 : ");      /*on demande l'opérande n°2 */
 scanf ("%d",&b);
 switch (operateur)
   { case '+': { result=a+b; break; }
     case '-': { result=a-b; break; }
     case '*': { result=a*b; break; }
     default : { printf ("\n\nl'op,rateur saisie n'est pas correct"); B=1;}
   }
 if (B==0) printf ("\n\nR,sultat : %d",result);

 getch();                           /* attente de validation */
 printf ("\n\n Voulez vous encore jouer (entrez 1 pour Oui)?");
 scanf ("%d",&encore);              /* on récupère la réponse */
 } while (encore==1);               /* si la réponse est 1, on relance la boucle */
}                                   /* fin */
```

Application 2: Compléter le programme C suivant :

afin qu'il affiche la somme de deux horaires saisis en début de traitement. Par exemple l'opération «23:30» + «2:30» donnera : «02:00».

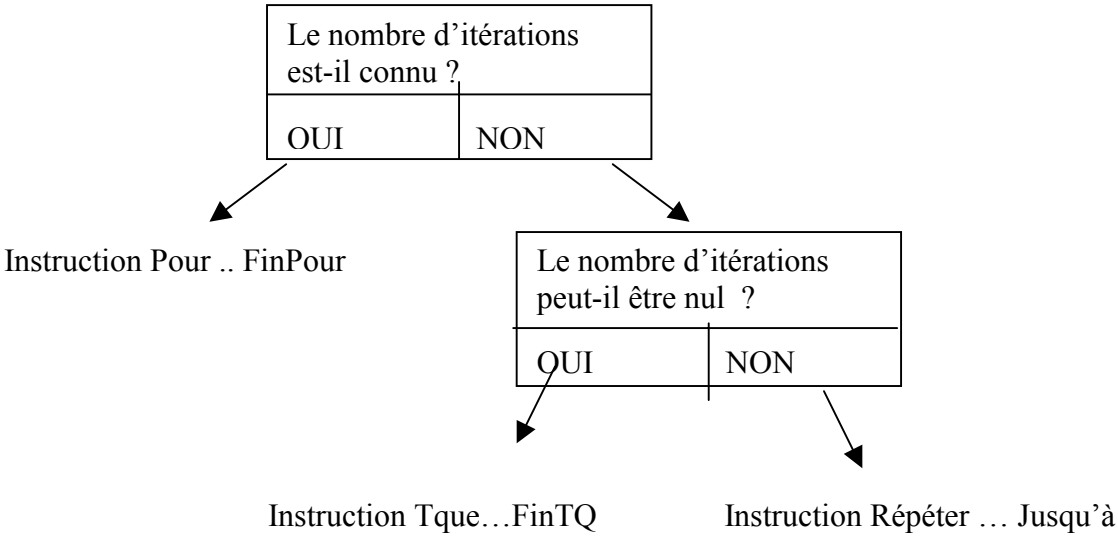
```
void main()
{
 struct horaire                  /* Déclaration d'un type date */
   {  int heure;                 /* 1ere case : entier jour */
      int minute;                /* 2ieme case : entier mois */
   };
 struct horaire H1,H2,H3;        /* D du type date contiendra la date saisie */
 int encore=0;

 do {
 clrscr();                       /* efface l'écran */
 printf ("\nhoraire nø1 : ");    /* Affiche l'interrogation */
 scanf ("%d:%d",&H1.heure,&H1.minute);  /* récupère la date saisie par l'utilisateur */
 printf ("\n\nhoraire nø2 : ");  /*on écrit l'année scolaire en cours */
 scanf ("%d:%d",&H2.heure,&H2.minute);
 H3.heure=H1.heure+H2.heure;
 H3.minute=H1.minute+H2.minute;
 if (H3.minute>=60) { H3.minute-=60;
                      H3.heure++;
                    }
 if (H3.heure>=24) H3.heure-=24;
 printf ("\n\nSomme des horaires : %d:%d",H3.heure,H3.minute);

 getch();                        /* attente de validation */
 printf ("\n\n Voulez vous encore jouer (entrez 1 pour Oui)?");
 scanf ("%d",&encore);           /* récupère la réponse */
 } while (encore==1);            /* si la réponse est 1, on relance la boucle */
}                                /* fin */
```

4.4) Les instructions itératives

On rappelle qu'en algorithmique, il existe plusieurs instructions pour mettre en œuvre un traitement itératif. Le choix entre ces différentes instructions est basé sur le nombre d'itérations comme le montre le schéma ci-dessous :



Instruction algorithmique	Traduction en langage C
TantQue Condition Faire Bloc d'instructions FinTQ	**while (**Condition**)** **{** Bloc d'instructions **}**
Répéter Bloc d'instructions Jusqu'à Condition	**do** **{** Bloc d'instructions **}** **while (!(**Condition**))**
Pour Compteur ← Valeur initiale à Valeur finale Bloc d'instructions FinPour	**For (**Compteur=Valeur initiale **;**Compteur <=Valeur finale **;** Compteur ++**)** **{** Bloc d'instructions **}**

Remarques :

- Si le bloc d'instructions est limité à une seule, on peut se dispenser des accolades.
- La condition spécifiée dans l'instruction « **do … while** » est la négation de la condition exprimée dans le « Répéter… jusqu'à ».
- L'exécution de l'instruction : **for(**expression1 ;expression2 ;expression3**)** **{**bloc d'instructions**}** est équivalente à la séquence d'instructions suivante :

  Evaluer expression1
  While (Expression2**)**
  { bloc d'instructions
  Evaluer expression3 **}**

Application 1: Traduire en langage C l'algorithme suivant :

qui est destiné à calculé le plus grand commun diviseur de deux nombres a et b saisis au clavier (méthode d'Euclide). On suppose que a est supérieur à b.

```
Variables
        a, b, r : Entier
Début
        Afficher (« Valeur de a : »)
        Entrer a
        Afficher (« Valeur de b : »)
        Entrer b
        r ← a mod b
        TantQue r <> 0 Faire
                a ← b
                b ← r
                r ← a mod  b
        FTQ
        Afficher «Le PGCD est : », b
Fin
```

Remarque : l'expression a mod b renvoie le reste de la division euclidienne de a par b.

```
#include <stdio.h>

void main()
{
  int a,b,r=1;

  clrscr();

  printf ("\n\nValeur de a : ");
  scanf ("%d",&a);
  printf ("\nValeur de b : ");
  scanf ("%d",&b);
  r=a%b;
  while (r!=0)
    { a=b;
      b=r;
      r=a%b;
    }
  printf ("\n\nLe PGCD est : %d",b);
  getch();
}
```

Application 2: Ecrire le programme C qui affiche le nombre de chiffres qui compose un entier positif saisi au clavier :

```
#include <stdio.h>

void main()
{
 int i=0,test=0;
 struct stat
   { int valeur;
     int effectif;
   };
 struct stat tstat[100];

 clrscr();
 do {
 printf ("\n\nEntrer la valeur de la case %d du tableau : ",i);
 scanf ("%d",&tstat.valeur[i]);
 printf ("\n\nEntrer l'effectif de la case %d du tableau : ",i);
 scanf ("%d",&tstat.effectif[i]

 while (entier>9)
   { i++;
     entier/=10;
   }
 printf ("\n\nNombre de chiffres : %ld",i);
 getch();
 i=1;
 printf ("\n\nVoulez vous continuer (Oui=1) :");
 scanf ("%ld",&test);
 } while (test==1);
```

Application 3: on considère le tableau d'enregistrements Tstat ci-dessous :

et destiné à contenir une série statistique simple.

```
Struct Stat
{         int Valeur;
          int Effectif;
};
struct stat Tstat [100];
```

Ecrire un programme C qui enchaîne les différents traitements suivants :

- valorisation du tableau Tstat par la saisie d'une liste de valeurs ; la fin de la saisie sera marquée par la valeur 999,
- affichage, pour chaque valeur différente saisie, de son effectif,
- affichage de la valeur qui admet le plus grand effectif. On suppose que cette valeur est unique.

Application 4: On considère le tableau à deux dimensions TD d'entiers :
(8 lignes, 8 colonnes) défini ci-dessous. Ecrire un programme C qui demande à l'utilisateur :
- la valeur d'un entier,
- le composant du tableau qu'il veut renseigner avec la valeur précédemment saisie en respectant la codification suivante :

T pour l'intégralité du tableau,
E pour une cellule du tableau ; dans ce cas l'utilisateur doit identifier le numéro de ligne et le numéro de colonne de la cellule,
L pour une ligne du tableau ; dans ce cas l'utilisateur doit identifier le numéro de ligne,
C pour une colonne du tableau ; dans ce cas l'utilisateur doit identifier le numéro de colonne,
D pour la diagonale : coin haut gauche, coin bas droit.

Un exemple d'exécution du programme est fourni ci-dessous (les valeurs saisies sont soulignées) :
Valeur ? 5
Composant ? L
N° de ligne ? 5

Pour simplifier l'algorithme, on ne devra pas envisager de contrôle sur les numéros de ligne et/ou de colonne saisis.

V. Les FONCTIONS et les BIBLIOTHEQUES

5.1) Utilisation et création de fonctions

Dans ce paragraphe, seuls les modules admettant:
- 0 à n paramètres d'entrée
- 0 ou 1 paramètre de sortie

seront traités et seront désignés par le terme de fonction. Cette terminologie diffère en algorithmique. Dans ce domaine, on appelle, en effet, *fonction* les modules qui admettent 1 et 1 seul paramètre de sortie, et *procédure* les autres modules

Une fonction est un outil destinée à faciliter le développement d'applications. Ainsi, le langage C fournit dans des bibliothèques, des fonctions pour répondre aux besoins de diverses applications. *Par exemple, la bibliothèque « math.h » regroupe différentes fonctions trigonométriques et logarithmiques*. Pour pouvoir utiliser les fonctions d'une bibliothèque il faut spécifier la clause #include <NomDeLaBibliothèque> en début de programme. *Le programme C ci-dessous utilise la fonction « sqrt » de la bibliothèque « math.h » pour calculer les solutions de l'équation du second degré* $ax^2+bx+c=0$.

```
#include <math.h>
void main()
```

```
{float a,b,c,delta ;
delta =b*b-4*a*c ;
if (delta < 0)
        printf (« pas de solution réelle\n ») ;
else
        if (delta == 0)
                printf (« la solution double est : %f\n »,-b/(2*a)) ;
        else
                {
                printf (« la première solution est : %f\n »,(-b-sqrt(delta))/(2*a)) ;
                printf (« la seconde solution est : %f \n», (-b+sqrt(delta)/(2*a)) ;
                }
}
```

Le programmeur peut aussi créer ses propres fonctions et ensuite les intégrer dans une bibliothèque afin de les mettre à la disposition d'autres programmeurs. En C, la création d'une fonction respecte la syntaxe suivante :

TypeRes NomFonction (Liste des paramètres d'entrée)
{ Déclaration des variables locales
Liste des instructions
}

Remarques :

- Chaque paramètre d'entrée est décrit par son type suivi de son nom ; les paramètres dans la liste sont séparés par une virgule.
- Dans le cas où la fonction ne renvoie aucun résultat, c'est le mot clé « **void** » qui est spécifié avant le nom de la fonction.
- Dans le cas où la fonction renvoie un résultat, c'est l'instruction « **return (**Expression**)** » qui permet de renvoyer au programme appelant le résultat souhaité.

Les exemples ci-dessous présentent deux fonctions : l'une chargée de renvoyer le périmètre d'un rectangle à partir des deux paramètres d'entrée représentant la longueur et la largeur, l'autre chargée d'afficher ce même périmètre.

```
int Perimetre(int Largeur, int longueur)
  {
 return (2*(Largeur+Longueur)) ;
  }
```

```
void AffichePerimetre (int Largeur, int Longueur)
  {
 printf (« le périmètre est : %d »,2*(Largeur+Longueur))
  }
```

Dans le programme appelant, il va de soi que l'appel de la fonction diffère totalement selon si la fonction renvoie 0 ou 1 résultat. Ainsi une fonction de type « **void** » doit s'utiliser comme une instruction, alors qu'une fonction qui renvoie un résultat doit être intégrée à une instruction algorithmique. Ce propos est illustré par les appels des deux fonctions présentées ci-dessus.

```
void main()
{
        printf(« le périmètre d'un rectangle de largeur 2 et de longueur 4 est  : %d »,
Perimetre(2,4)) ;
        AffichePerimetre(2, 4) ;
}
```

Application 1: Ecrire la fonction Cqui calcule et renvoie le PGCD :

de deux entiers positifs a et b. On suppose que a est supérieur à b.

```
int pgcd (int a, int b)
{ int r;
  r=a%b;
  while (r!=0) {a=b; b=r; r=a%b;}
  return (b);
}
```

Application 2: fonction qui calcule l'arrondi au ½ point supérieur d'une valeur :

```
float arrondi (float note)
{
 int notea,i;
 i = note - ( int ) note;
 notea = (i = = 0 ) ? note : ( i > 0.5 ) ? ( int ) note + 1 : ( int ) note + 0.5;
 return (notea);
}
```

Application 3: La bibliothèque « date.h » :

propose différentes fonctions permettant de manipuler des dates. Parmi celles-ci on trouve les deux fonctions suivantes :

int numerojour(struct date d) fonction qui renvoie le numéro du jour dans l'année associé à la date d
struct date converdate(int q) fonction qui renvoie la date associée au quantième q
int nombrejours (int a) fonction qui renvoie le nombre de jours de l'année a

date est un type défini ci-dessous :

```
struct date
        {
        int jour ;
        int mois ;
        int annee ;
        };
```

a) Ecrire le programme C qui affiche la date du lendemain d'une date saisie au clavier
b) Transformer ce programme en une fonction qui calcule la date du lendemain

5.2) La fonction main()

Tout programme écrit en langage C doit comporter une fonction **main()** qui est le point d'entrée de l'application. Cela signifie que lors du lancement de l'application depuis l'environnement du système d'exploitation (DOS, ou UNIX), l'ordinateur exécutera les instructions contenues dans la fonction « main() ». *Par exemple, la saisie sous DOS de la commande « pgcd » déclenchera l'exécution du contenu de la fonction « main() » associée à ce programme.*

Pour permettre à l'utilisateur de mentionner des paramètres au niveau de la commande de lancement de son application on doit mentionner dans l'en-tête de la fonction main() les paramètres « **argc** » et « **argv** ». Le paramètre « argc » est un entier donnant le nombre d'arguments de la ligne de commande passés à « main() ». Le paramètre « argc » est un tableau de chaînes de caractères contenant toutes les valeurs des arguments passés à « main() ». *Le programme ci-dessous présente la fonction main() associé au calcul du PGCD de deux nombres a et b avec a et b non plus saisis lors de l'exécution mais valorisés dans la commande de lancement. On suppose que la bibliothèque « arith.h » contient la fonction PGCD.*

```
#include <arith.h>
void main(int argc, char * argv[])
{if (argc != 3)
        printf (« erreur sur le nombre d'arguments ») ;
else
        printf (« la valeur du PGCD de %d et %d est : %d », a, b,
PGCD(atoi(argv[1]),atoi(argv[2]))) ;
}
```

Remarques :

- Le nom de la commande est considéré comme un argument. ,
- La fonction «atoi» transforme une chaîne de caractères représentant un entier en un entier.

Pour calculer le PGCD de 24 et 16, l'utilisateur devra saisir, depuis l'environnement du système d'exploitation la commande : pgcd 24 16

Lors du lancement de l'application, les arguments sont séparés par des espaces. Dans le cas où un argument intègre lui-même des espaces, il est nécessaire de l'encadrer par des guillemets. La fonction main() peut retourner un résultat qui est un code destiné à réaliser un compte-rendu sur l'exécution du programme. Si dans le programme, l'instruction « **exit(**numéro**)** » est référencée, c'est le numéro fourni en argument de l'instruction « **exit** » qui est retourné. Pour exploiter le résultat retourné par la fonction main, il faut spécifier l'entête de la fonction main() de la manière suivante :

int main(….)

Travaux pratiques

1. On souhaite construire la bibliothèque « fraction.h » destinée à réaliser les opérations suivantes sur les fractions :
   - initialiser une fraction à partir de la donnée d'un numérateur et d'un dénominateur
   - afficher une fraction sous la forme : numérateur/dénominateur
   - ajouter deux fractions
   - multiplier deux fractions
   - diviser deux fractions
   - calculer l'inverse d'une fraction
   - mettre sous forme irréductible une fraction

a) Proposer une représentation interne pour implanter en mémoire centrale une fraction.
b) Définir en C, pour chacune des opérations citées, l'interface de la fonction chargée de la réaliser.
c) Ecrire en langage C le contenu de chacune des fonctions. Les tester sur machine.

2. On souhaite construire la bibliothèque « periode.h » destinée à réaliser les opérations suivantes sur des périodes horaires caractérisées par un horaire début (exprimé en heure minute) inférieur à un horaire fin (exprimé en heure minute).
   - initialiser une période à partir de la donnée d'un horaire début et d'un horaire fin. On suppose que l'horaire début est inférieur à l'horaire fin
   - afficher une période sous la forme [hh1 :mm1 ;hh2 :mm2]
   - calculer l'intersection entre deux périodes ; dans le cas où la période résultat est vide, la période [00 :00 ;00 :00] sera renvoyée.
   - Calculer le nombre de minutes associée à une période

a) Proposer une représentation interne pour implanter en mémoire centrale une période ainsi définie.
b) Définir en C, pour chacune des opérations citées, l'interface de la fonction chargée de la réaliser.
c) Ecrire en C le contenu de chacune des fonctions. Les tester sur machine.

VI. ANNEXES

6.1) Calcul des périmètres et surface d'un rectangle

```
#include <stdio.h>

void main()
{
  float surface, perimetre, longueur, largeur;
  clrscr();
  printf ("Saisir la longueur: ");
  scanf ("%f",&longueur);
  printf ("\nSaisir la largeur: ");
  scanf ("%f",&largeur);
  perimetre=2*(longueur+largeur);
  surface=longueur*largeur;
  printf("\n\nle p,rimŠtre est %f et la surface est %f",perimetre,surface);
  getch();
}
```

6.2) Calcul d'une note arrondie au ½ point

```
#include <stdio.h>

void main()
{
  float note,notea,i;
  clrscr();
  printf ("Saisir la note: ");
  scanf ("%f",&note);
  i=note-(int)note;
  notea=(i==0)?note:(i>0.5)?(int)note+1:(int)note+0.5;
  printf ("\nLa note arrondie est %f",notea);
  getch();
}
```

6.3) Vérification de la majorité

```
#include <stdio.h>

void main()
{
 struct date
   {  int jour;
      int mois;
      int annee;
   };
 struct date N,J;
 int an,mois,jour,test=0;

 clrscr();
 printf ("Date de naissance : ");
 scanf ("%d/%d/%d",&N.jour,&N.mois,&N.annee);
 printf ("\nDate du jour : ");
 scanf ("%d/%d/%d",&J.jour,&J.mois,&J.annee);
 an=J.annee-N.annee;
 if (an>18) test=1;
  else { if (an<18) test=0;
       else { mois=J.mois-N.mois;
              if (mois>0) test=1;
              else { if (mois<0) test=0;
                     else { jour=J.jour-N.jour;
                            if (jour>=0) { test=1;
                                           if (jour==0) printf ("\n\nBON ANNIVERSAIRE!!!");
                                           }
                            else test=0;
                          }
                   }
            }
       }

 if (test==0) printf ("\n\nVous ^tes mineur");
  else printf ("\n\nVous ^tes majeur");
 getch();
}
```

6.4) Affichage de l'année scolaire en cours

```
#include <stdio.h>

void main()
{
 struct date          /* Déclaration d'un type date */
   {  int jour;       /* 1ere case : entier jour */
      int mois;       /* 2ieme case : entier mois */
      int annee;      /* 3ieme case : entier annee */
   };
 struct date D;       /* D du type date contiendra la date saisie */
 int an=0,encore=0;   /* entier an contiendra l'année ajustee */
                      /* booléen encore pour boucle de relance du prog */
 do {                 /* lance la boucle de relance du prog */
 clrscr();            /* efface l'écran */
 printf ("\nDate du jour ? ");   /* Affiche l'interrogation */
 scanf ("%d/%d/%d",&D.jour,&D.mois,&D.annee);  /* récup. date saisie par l'utilisateur*/
 if (D.mois>8) {              /* teste si le mois est postérieur à aout */
              an=D.annee+1; /* si oui, an recoit annee + 1 */
              printf ("\n\nAnn,e scolaire : %d-%d",D.annee,an); /*écrit année scol en cours*/
              }
   else { an=D.annee-1;    /* sinon, an recoit annee - 1 */
        printf ("\n\nAnn,e scolaire : %d-%d",an,D.annee);   /* on ecrit l'année en cours */
       }
 getch();                        /* attente de validation */
 printf ("\n\n Voulez vous encore jouer (entrez 1 pour Oui)?");  /* demande si l'on veut
relancer le programme */
 scanf ("%d",&encore);   /* récupère la réponse */
 } while (encore==1);    /* si la réponse est 1, on relance la boucle */
}                    /* fin */
```

6.5) Somme de deux horaires

```
#include <stdio.h>                  /* Appel à la bibliothèque stdio */

void main()
{
 struct horaire                     /* Déclaration d'un type date */
   {  int heure;                    /* 1ere case : entier jour */
      int minute;                   /* 2ieme case : entier mois */
   };
 struct horaire H1,H2,H3;           /* D du type date contiendra la date saisie */
 int encore=0;

 do {
 clrscr();                          /* efface l'écran */
 printf ("\nhoraire nø1 : ");       /* Affiche l'interrogation */
 scanf ("%d:%d",&H1.heure,&H1.minute);  /* récupère la date saisie par l'utilisateur */
 printf ("\n\nhoraire nø2 : ");     /*on écrit l'année scolaire en cours */
 scanf ("%d:%d",&H2.heure,&H2.minute);
 H3.heure=H1.heure+H2.heure;
 H3.minute=H1.minute+H2.minute;
 if (H3.minute>=60) { H3.minute-=60;
                      H3.heure++;
                    }
 if (H3.heure>=24) H3.heure-=24;
 printf ("\n\nSomme des horaires : %d:%d",H3.heure,H3.minute);

 getch();                           /* attente de validation */
 printf ("\n\n Voulez vous encore jouer (entrez 1 pour Oui)?");
 scanf ("%d",&encore);              /* récupère la réponse */
 } while (encore==1);               /* si la réponse est 1, on relance la boucle */
}                                   /* fin */
```

6.6) Opérations mathématiques simples

```
#include <stdio.h>      /* Appel à la bibliothèque stdio */

void main()
{
 int a,b,result,B=0,encore=0;
 char operateur;

 do {
 clrscr();                          /* efface l'écran */
 printf ("\nop,rande nø1 : ");      /* Affiche l'interrogation */
 scanf ("%d",&a);                   /* récupère l'opérande date saisie par l'utilisateur */
 printf ("\nop,rateur (+,*,-) : ");
 scanf ("%s",&operateur);
 printf ("\nop,rande nø2 : ");      /*on demande l'opérande n°2 */
 scanf ("%d",&b);
 switch (operateur)
   { case '+': { result=a+b; break; }
     case '-': { result=a-b; break; }
     case '*': { result=a*b; break; }
     default : { printf ("\n\nl'op,rateur saisie n'est pas correct"); B=1;}
   }
 if (B==0) printf ("\n\nR,sultat : %d",result);

 getch();                           /* attente de validation */
 printf ("\n\n Voulez vous encore jouer (entrez 1 pour Oui)?");
 scanf ("%d",&encore);              /* on récupère la réponse */
 } while (encore==1);               /* si la réponse est 1, on relance la boucle */
}                                   /* fin */
```

6.7) PGCD par la division EUCLIDIENNE

```
#include <stdio.h>

void main()
{
  int a,b,r=1;

  clrscr();
  printf ("\n\nValeur de a : ");
  scanf ("%d",&a);
  printf ("\nValeur de b : ");
  scanf ("%d",&b);
  r=a%b;
  while (r!=0)
    { a=b;
      b=r;
      r=a%b;
    }
  printf ("\n\nLe PGCD est : %d",b);
  getch();
}
```

6.8) Connaître le nombre de chiffre d'un entier

```
#include <stdio.h>

void main()
{
 int i=0,test=0;
 struct stat
   { int valeur;
     int effectif;
   };
 struct stat tstat[100];

 clrscr();
 do {
 printf ("\n\nEntrer la valeur de la case %d du tableau : ",i);
 scanf ("%d",&tstat.valeur[i]);
 printf ("\n\nEntrer l'effectif de la case %d du tableau : ",i);
 scanf ("%d",&tstat.effectif[i]
 while (entier>9)
   { i++;
     entier/=10;
   }
 printf ("\n\nNombre de chiffres : %ld",i);
 getch();
 i=1;
 printf ("\n\nVoulez vous continuer (Oui=1) :");
 scanf ("%ld",&test);
 } while (test==1);
```

6.9) Test des fonctions PGCD et arrondi

```
#include <stdio.h>

float arrondi(float note)
{
 int notea,i;
 i=note-(int)note;
 notea=(i==0)?note:(i>0.5)?(int)note+1:(int)note+0.5;
 return (notea);
}

int pgcd (int a, int b)
{ int r;
 r=a%b;
 while (r!=0) {a=b; b=r; r=a%b;}
 return (b);
}

void main()
{
 int var1,var2,test=1;
 float var3;
 clrscr();
 do
 {  clrscr();
    printf ("\nSaisir les deux nombres : ");
    scanf ("%d %d",&var1,&var2);
    printf ("\nle PGCD est %d",pgcd(var1,var2));
    getch();
    printf ("\nVoulez vous continuer (Non=0)? ");
    scanf ("%d",&test);
 } while (test!=0);
 test=1;
 do
 {  clrscr();
    printf ("\Saisir la note : ");
    scanf ("%f",&var3);
    printf ("\La note arrondie est %f",arrondi(var3));
    getch();
    printf ("\nVoulez vous continuer (Non=0)? ");
    scanf ("%d",&test);
  } while (test!=0);
}
```