

LES STRUCTURES DE CONTROLES ET LES ENTREES ET SORTIES

I. INTRODUCTION

Une variable est une zone mémoire destinée à stocker des données lors du traitement. Elle est caractérisée par un nom, un type et une valeur.

Déclaration d'une variable : **type nom_variable=valeur_initiale ;**

Exemples :

- ✓ **int a=0 ;**
- ✓ **int b =10;**
- ✓ **int c=-45 ;**
- ✓ **int a=0, b=10, c=-45 ;**

Une constante est une valeur qui ne varie pas au cours du traitement.

Exemple :

- ✓ **const float pi=3,14 ;**
- ✓ **#define pi 3,14;**

II. LES FONCTIONS D'ENTREES ET SORTIES

Il s'agit des fonctions de la librairie standard `stdio.h` utilisées avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'écran. Sur certains compilateurs, l'appel à la librairie `stdio.h` par la directive au préprocesseur

`#include <stdio.h>`

n'est pas nécessaire pour utiliser `printf` et `scanf`.

II.1 . LA FONCTION D'AFFICHAGE :PRINTF

La fonction **printf** est une fonction d'impression formatée, ce qui signifie que les données sont converties selon le format particulier choisi.

Sa syntaxe est `printf("chaîne",expression ou variable);`

chaîne contient le texte à afficher et les spécifications de format correspondant à chaque expression de la liste. Les spécifications de format ont pour but d'annoncer le format des données à visualiser.

Elles sont introduites par le caractère %, suivi d'un caractère désignant le format d'impression. Les formats d'impression en C sont donnés dans la table 1 suivante.

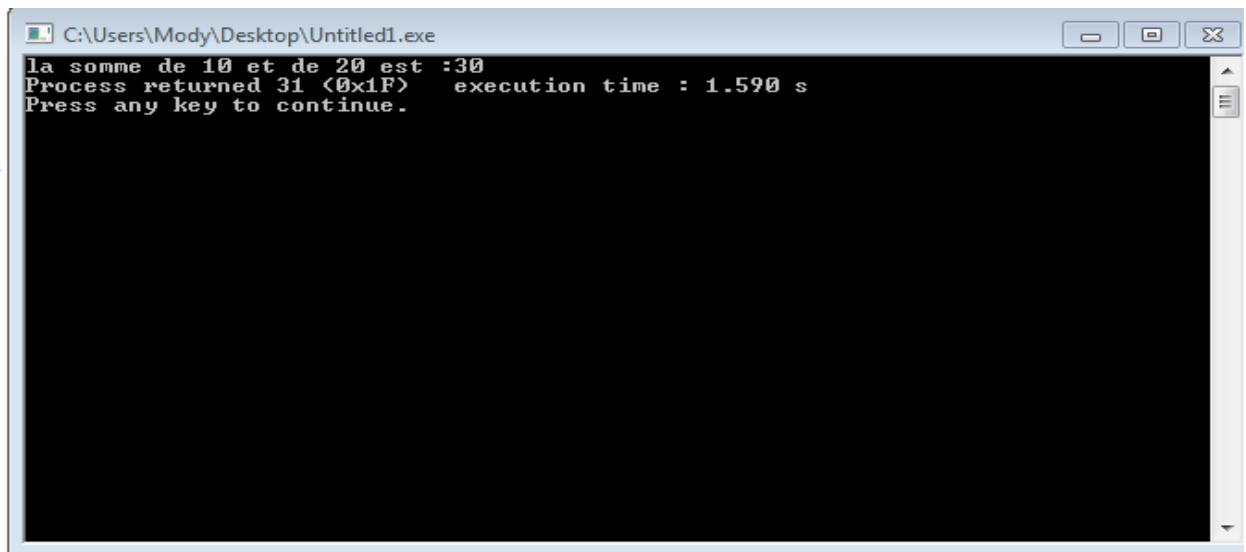
Format	conversion en	écriture
%d	int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	unsigned int	octale non signée
%lo	unsigned long int	octale non signée
%x	unsigned int	hexadécimale non signée
%lx	unsigned long int	hexadécimale non signée
%f	double	décimale virgule fixe
%lf	long double	décimale virgule fixe
%e	double	décimale notation exponentielle
%le	long double	décimale notation exponentielle
%g	double	décimale, représentation la plus courte parmi %f et %e
%lg	long double	décimale, représentation la plus courte parmi %lf et %le
%c	unsigned char	caractère
%s	char*	chaîne de caractères

Table 1: Formats d'impression pour la fonction printf

Exemple :

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a=0,b=20,c ;
    c=a+b ;
    printf("la somme de %d et de %d est :%d",a,b,c) ;
}
```

Après exécution on obtient :



La fonction **printf** permet d'imprimer une chaîne de caractère à l'écran.

II.2 LA FONCTION D'ECRITURE SCANF

La fonction Scanf permet de saisir des données au clavier et de les stocker aux adresses spécifiées par les arguments de la fonctions. **scanf("FORMAT",&VARIABLE) ;**

La chaîne de contrôle indique le format dans lequel les données lues sont converties. Elle ne contient pas d'autres caractères (notamment pas de \n). Comme pour printf, les conversions de format sont spécifiées par un caractère précédé du signe % . Les formats valides pour la fonction scanf diffèrent légèrement de ceux de la fonction printf et sont donnés dans la table 2.

Les données à entrer au clavier doivent être séparées par des blancs ou des <RETURN> sauf s'il s'agit de caractères. On peut toutefois fixer le nombre de caractères de la donnée à lire. Par exemple %3s pour une chaîne de 3 caractères, %10d pour un entier qui s'étend sur 10 chiffres, signe inclus.

Exemple

Saisir deux entiers au clavier puis calculer le quotient a/b.

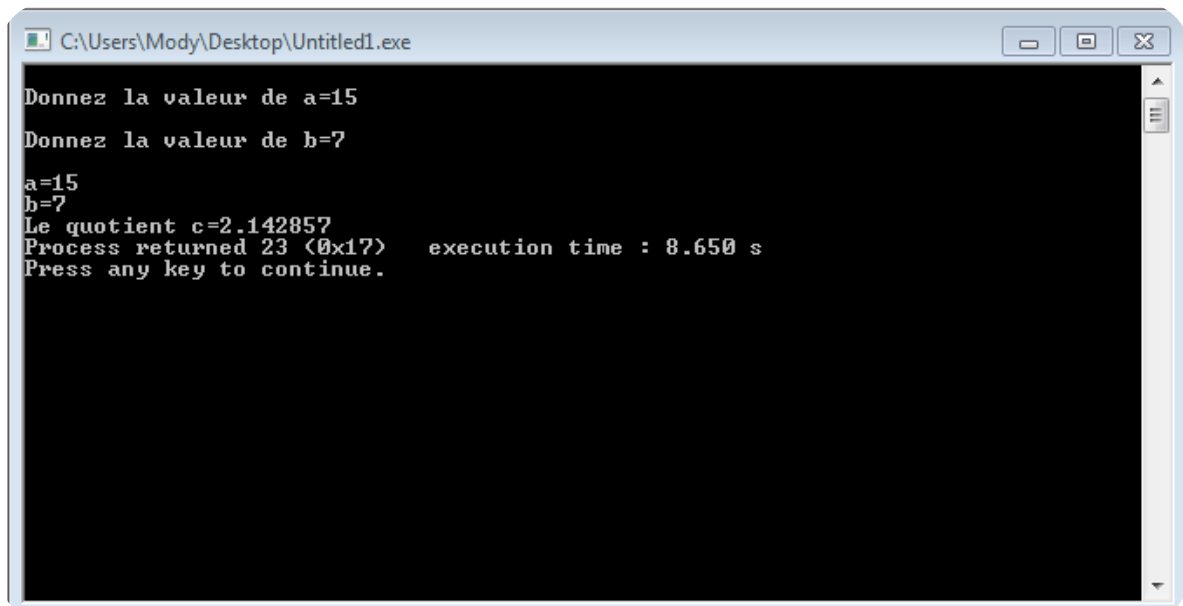
```
#include<stdio.h>
#include<conio.h>
main()
{
    int a=0,b=0;
    float c=0 ;
    printf("\nDonnez la valeur de a=");
    scanf("%d",&a);
    printf("\nDonnez la valeur de b=");
    scanf("%d",&b);
```

```

        c=(float)a/b;
        printf("\na=%d",a);
        printf("\nb=%d",b);
        printf("\nLe quotient c=%f",c);
    }

```

L'exécution de ce programme donne le résultat ci-dessous :



```

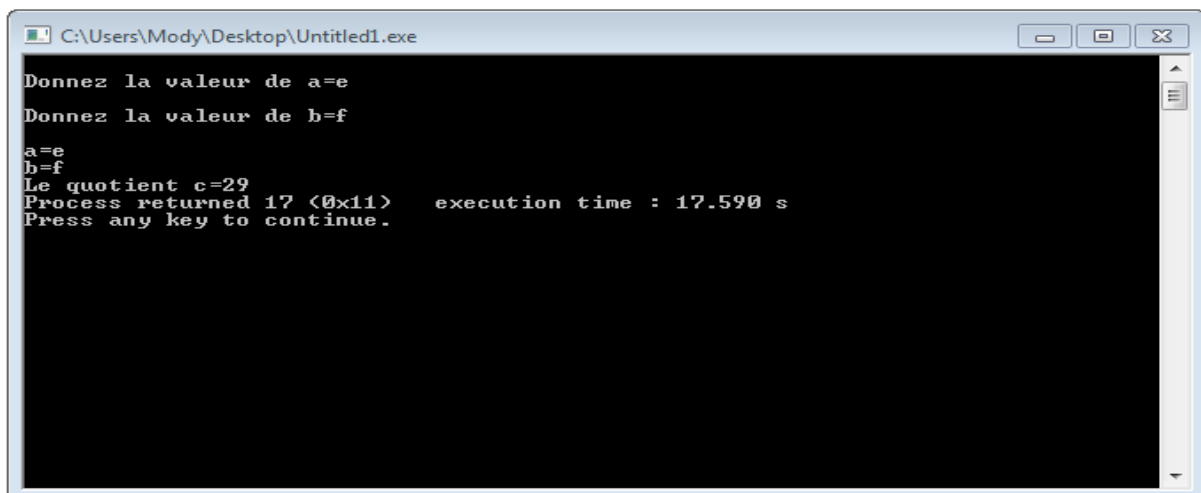
C:\Users\Mody\Desktop\Untitled1.exe
Donnez la valeur de a=15
Donnez la valeur de b=7
a=15
b=7
Le quotient c=2.142857
Process returned 23 (0x17)    execution time : 8.650 s
Press any key to continue.

```

```

#include<stdio.h>
#include<conio.h>
main()
{
    int a=0,b=0;
    int c=0 ;
    printf("\nDonnez la valeur de a=");
    scanf("%x",&a);
    printf("\nDonnez la valeur de b=");
    scanf("%x",&b);
    c=a+b;
    printf("\na=%x",a);
    printf("\nb=%x",b);
    printf("\nLe quotient c=%d",c);
}

```



```

C:\Users\Mody\Desktop\Untitled1.exe
Donnez la valeur de a=e
Donnez la valeur de b=f
a=e
b=f
Le quotient c=29
Process returned 17 (0x11)    execution time : 17.590 s
Press any key to continue.

```

format	type d'objet pointé	représentation de la donnée saisie
%d	int	décimale signée
%hd	short int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%hu	unsigned short int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	int	octale
%ho	short int	octale
%lo	long int	octale
%x	int	hexadécimale
%hx	short int	hexadécimale
%lx	long int	hexadécimale
%f	float	flottante virgule fixe
%lf	double	flottante virgule fixe
%Lf	long double	flottante virgule fixe
%e	float	flottante notation exponentielle
%le	double	flottante notation exponentielle
%Le	long double	flottante notation exponentielle
%g	float	flottante virgule fixe ou notation exponentielle
%lg	double	flottante virgule fixe ou notation exponentielle
%Lg	long double	flottante virgule fixe ou notation exponentielle
%c	char	caractère
%s	char*	chaîne de caractères

Table 2: Formats de saisie pour la fonction scanf

II.3 IMPRESSION ET LECTURE DE CARACTERES

Les fonctions getchar et putchar permettent respectivement de lire et d'imprimer des caractères. Il s'agit de fonctions d'entrées-sorties non formatées.

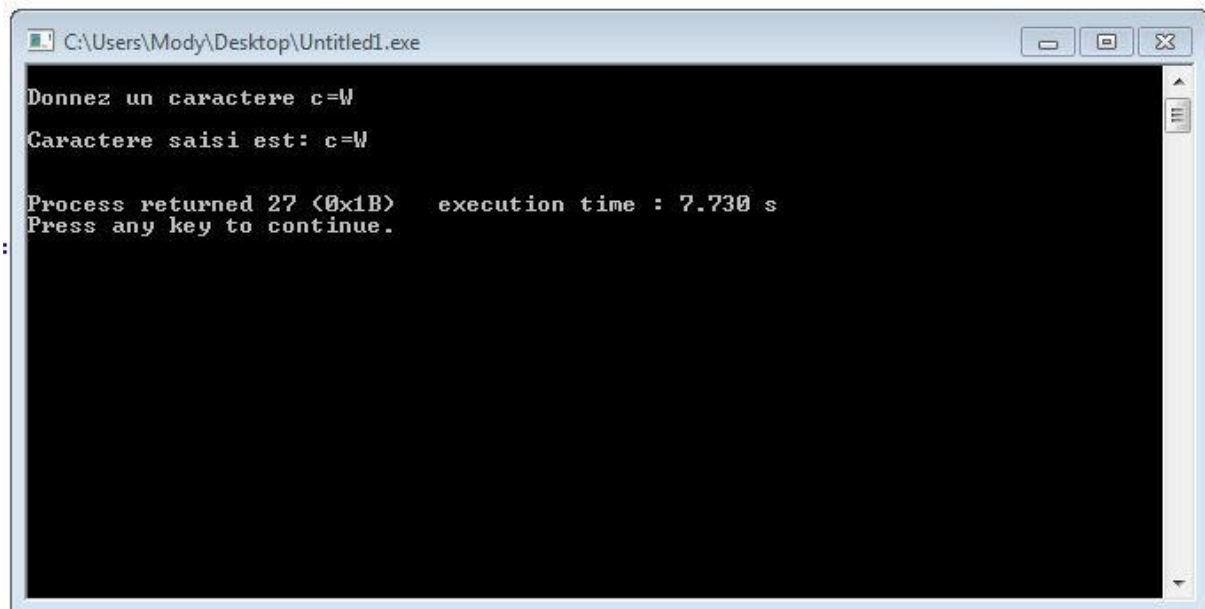
La fonction getchar retourne un int correspondant au caractère lu. Pour mettre le caractère lu dans une variable caractere, on écrit :

caractere = getchar(); \longleftrightarrow **scanf("%c",&c) ;**

La fonction putchar écrit caractere sur la sortie standard :

putchar(caractere); \longleftrightarrow **printf("%c",c) ;**

```
#include<stdio.h>
#include<conio.h>
main()
{
    char c;
    printf("\nDonnez un caractere c=");
    c=getchar();
    printf("\nCaractere saisi est: c=%c",c);
}
```



```
C:\Users\Mody\Desktop\Untitled1.exe
Donnez un caractere c=W
Caractere saisi est: c=W
Process returned 27 (0x1B) execution time : 7.730 s
Press any key to continue.
```

III. LES STRUCTURES CONDITIONNELLES

On appelle instruction de contrôle toute instruction qui permet de contrôler le fonctionnement d'un programme. Parmi les instructions de contrôle, on distingue les instructions de branchement et les boucles. Les instructions de branchement permettent de déterminer quelles instructions seront exécutées et dans quel ordre.

III.1 BRANCHEMENT CONDITIONNEL « if---else » (SI ... SINON)

La forme la plus générale est celle-ci :

```
if (CONDITION )
BLOC1
else
BLOC2
```

NB : Dans un **if** il peut y'arriver qu'on ait un autre **if** et dans ce cas on parle de if imbriqué

```
if (CONDITION1 )
    bloc1
else
    if (CONDITION2 )
        bloc2
    else
        bloc3
```

Exemple :

Ecrire un programme qui permet de saisir deux entier a et b puis calcule :

La différence a-b si a est supérieur ou égal à b

La différence b-a si a est inférieur à b

```

#include<stdio.h>
#include<conio.h>
main()
{
    int a,b,difference;
    printf("Donnez la valeur de a=");
    scanf("%d",&a);
    printf("Donnez la valeur de b=");
    scanf("%d",&b);
    if(a>=b)
        difference=a-b;
    else
        difference=b-a;
    printf("\n\nLa difference est:%d\n\n",difference);
}

```

```

C:\Users\Mody\Desktop\Untitled1.exe
Donnez la valeur de a=100
Donnez la valeur de b=-95

La difference est:195

Process returned 25 (0x19)   execution time : 11.780 s
Press any key to continue.

```

Avec un nombre quelconque de **else if (condition)**. Le dernier **else** est toujours facultatif. La forme la plus simple est :

```

if (condition)
instruction

```

III.2 BRANCHEMENT MULTIPLE « switch »

Sa forme la plus générale est celle-ci :

```

switch (expression)
{
    case constante1:
    liste d'instructions 1
    break;
    case constante2:
    liste d'instructions 2
    break;

    case constanteN:
    liste d'instructions N
    break;
    default:
    liste d'instructions M
}

```

```

    break;
}

#include<stdio.h>
#include<conio.h>
main()
{
    int a,b,resultat;
    char operation;
    float resultat1;
    printf(" + =====> Addition\n");
    printf(" - =====> Soustraction\n");
    printf(" x =====> Multiplication\n");
    printf(" / =====> Division\n");
    printf("Saisir l'operation que vous voulez effectuer =");
    operation=getchar();
    printf("Donnez la valeur de a=");
    scanf("%d",&a);
    printf("Donnez la valeur de b=");
    scanf("%d",&b);
    switch(operation)
    {
        case '+':
            resultat=a+b;
            printf("La somme est :%d",resultat);
            break;
        case '-':
            resultat=a-b;
            printf("La différence est :%d",resultat);
            break;
        case 'x':
            resultat=a*b;
            printf("Le produit est :%d",resultat);
            break;
        case '/':
            resultat1=(float)a/b;
            printf("Le quotient est :%f",resultat1);
            break;
        default:
            printf("OPERATION INCONNUE");
            break;
    }
}

```

Si la valeur de l'expression est égale à l'une des constantes, la liste d'instructions correspondant est exécutée.

Sinon la liste d'instructions M correspondant à default est exécutée. L'instruction default est facultative.

III.3. LES BOUCLES

Les boucles permettent de répéter une série d'instructions.

III.3.1 BOUCLE « while »

La syntaxe de while est la suivante :

WHILE (CONDITION)

BLOC INSTRUCTIONS

Tant que la condition est vérifiée, le bloc d'instructions est exécuté. Si la condition est fausse au départ, instruction ne sera jamais exécutée.

Par exemple, le programme suivant imprime les entiers de 1 à 9.

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i=1;
    while(i<10)
    {
        printf("%d\t",i);
        i++;
    }
}
```

III .3.2. BOUCLE « do---while »

Il peut arriver que l'on ne veuille effectuer le test de répétition qu'après avoir exécuté l'instruction.

Dans ce cas, on utilise la boucle **do---while**.

Sa syntaxe est :

do

instruction

while (condition);

Ici, instruction sera exécutée tant que condition est vraie. Cela signifie donc que instruction est toujours exécutée au moins une fois.

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i=1;
    do
    {
        printf("%d\t",i);
        i++;
    }
    while(i<10);
}
```


III .3.3. BOUCLE « for » (pour)

La boucle for nous permet de répéter un traitement pour un nombre fini d'objets ou d'individus l'avantage de la boucle for est que nombre de répétition est connu dès le début du traitement.

La boucle for est caractérisée par une valeur de début une valeur de fin et un pas qui permet de se déplacer de la valeur de début à la valeur de fin.

Syntaxe :

for(expression1 ;condition ;expression 2)

```
{  
    Bloc d'instruction ;  
}  
#include<stdio.h>  
#include<conio.h>  
main()  
{  
    int i;  
    for(i=1;i<10;i++)  
    {  
        printf("%d\t",i);  
    }  
}
```

III.4. LES INSTRUCTIONS DE BRANCHEMENT NON CONDITIONNEL

III.4.1. BRANCHEMENT NON CONDITIONNEL « break »

On a vu le rôle de l'instruction break ; au sein d'une instruction de branchement multiple switch. L'instruction break peut, plus généralement, être employée à l'intérieur de n'importe quelle boucle. Elle permet d'interrompre le déroulement de la boucle, et passe à la première instruction qui suit la boucle. En cas de boucles imbriquées, break fait sortir de la boucle la plus intérieure.

Par exemple, le programme suivant :

```
main()  
{  
    int i;  
    for (i = 0; i < 5; i++)  
    {  
        printf("i = %d\n",i);  
        if (i == 3)  
            break;  
    }  
    printf("valeur de i a la sortie de la boucle = %d\n",i);  
}
```

Après exécution on obtient :



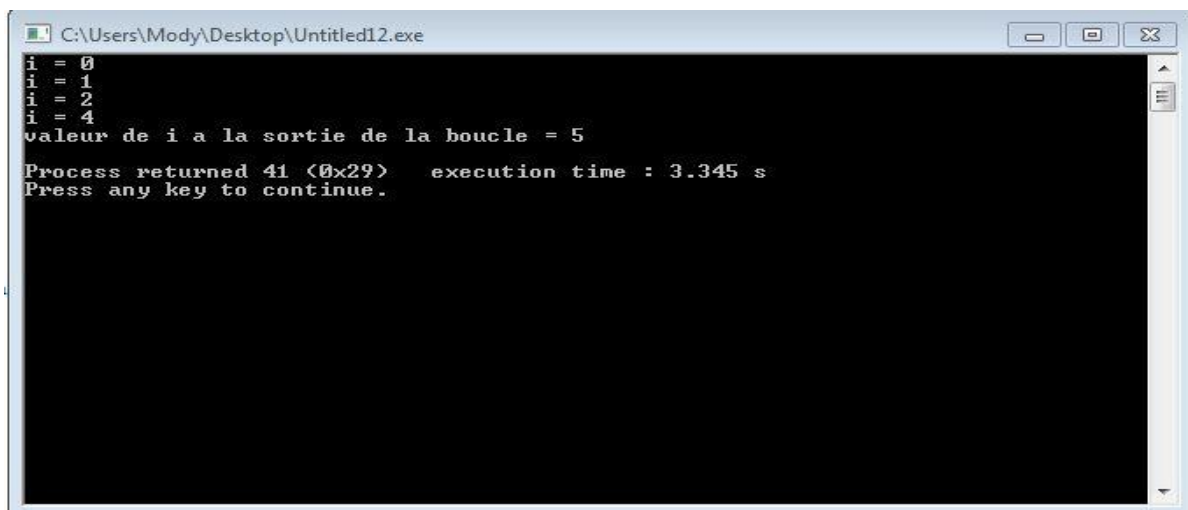
```
i = 0
i = 1
i = 2
i = 3
valeur de i a la sortie de la boucle = 3
Process returned 41 (0x29)    execution time : 3.332 s
Press any key to continue.
```

La valeur de i à la sortie de la boucle = 3.

III.4.2. BRANCHEMENT NON CONDITIONNEL « continue »

L'instruction continue permet de passer directement au tour de boucle suivant, sans exécuter les autres instructions de la boucle. Ainsi le programme

```
main()
{
int i;
for (i = 0; i < 5; i++)
{
if (i == 3)
continue;
printf("i = %d\n",i);
}
printf("valeur de i a la sortie de la boucle = %d\n",i);
}
```



```
i = 0
i = 1
i = 2
i = 4
valeur de i a la sortie de la boucle = 5
Process returned 41 (0x29)    execution time : 3.345 s
Press any key to continue.
```

III.4.3. BRANCHEMENT NON CONDITIONNEL « goto: »

Elle permet le branchement inconditionnel vers une instruction spécifiée par une étiquette dans le programme.

```

#include<stdio.h>
main()
{
    int a,b,c;
    scanf("%d",&a);
    scanf("%d",&b);
    etiquette: //on définit l'étiquète
    c=a%b;
    if(c!=0)
    {
        a=b;
        b=c;
        goto etiquette; // aller à l'étiquète
    }
    else
        c=b;
    printf("Le PGDC est %d",b);
}

```