

## CHAPITRE N° 4

# AUTRES CONTRÔLES DES PROGRAMMES

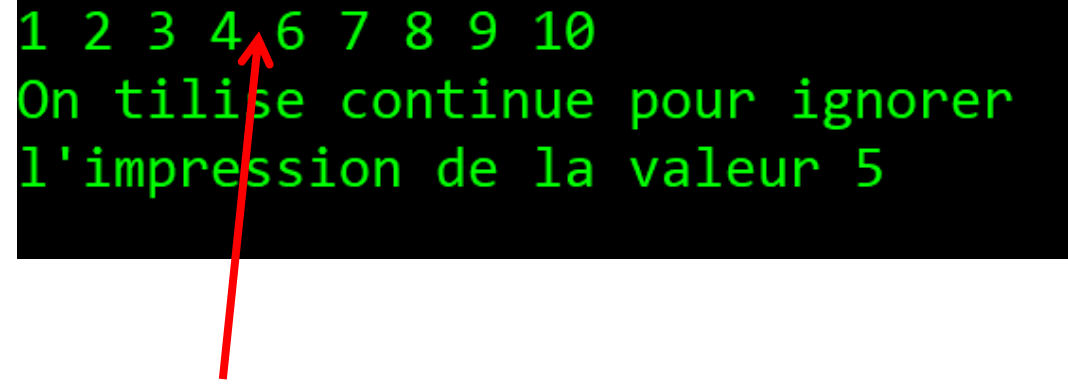
# Mot clé = continue

- **continue** force la prochaine itération à avoir lieu immédiatement, en ignorant toutes les instructions qui pourraient le suivre.
- L'instruction **continue** ne peut être utilisée qu'à l'intérieur d'une boucle (**for**, **do-while** et **while**) et non à l'intérieur d'une sélection **switch-case**.
- Lorsqu'elle est exécutée, elle transfère le contrôle à la condition (la partie de l'expression) dans une boucle **while** ou **do-while**, et à l'expression d'incrément dans une boucle **for**.
- Contrairement à l'instruction **break**, **continue** ne force pas la fin d'une boucle, il transfère simplement le contrôle à l'itération suivante.

## instructionContinue.c

```
#include <stdio.h>
void main()
{
    int iNum;
    for(iNum = 1; iNum <= 10; iNum++)
    {
        // sauter le code restant dans la boucle uniquement si iNum == 5
        if(iNum == 5)
            continue;
        printf("%d ", iNum);
    }
    printf("\nOn tilise continue pour ignorer l'impression de la valeur 5\n");
}
```

## Sortie :



```
1 2 3 4 6 7 8 9 10
On tilise continue pour ignorer
l'impression de la valeur 5
```

## Mot clé = goto

- L'instruction **goto** est une instruction de saut en C.
- Lorsque l'exécution du programme rencontre une instruction goto, l'exécution saute immédiatement, ou se branche, à l'emplacement spécifié par l'instruction **goto**.
- L'instruction est inconditionnelle car l'exécution se branche toujours lorsqu'une instruction **goto** est rencontrée, le branchement ne dépend d'aucune condition.
- Une instruction **goto** et son **libellé cible** doivent être situés dans la même fonction, bien qu'ils puissent être dans des blocs différents.
- Utilisez **goto** pour transférer l'exécution à la fois dans et hors de la boucle.
- Cependant, l'utilisation de l'instruction **goto** n'est pas recommandée.
- Utilisez toujours d'autres instructions de branchement en C.
- Lorsque l'exécution du programme se branche avec une instruction **goto**, aucun enregistrement (position) n'est conservé de l'origine de l'exécution.

# Exemple goto: calcul du plus petit commun diviseur

instructionGoto.c

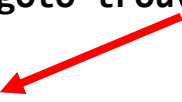
```
#include <stdio.h>
void main()
{
    int a;
    int b;

    printf("Entrez deux nombres : ");
    scanf("%d %d", &a, &b);

    int min = (a < b) ? a : b;
    int i;

    for (i = 2; i <= min; ++i)
        if (a % i == 0 && b % i == 0)
            goto trouve;

    trouve :
        printf("le plus petit diviseur de %d et %d est %d\n", a, b, i);
}
```



Sortie :

```
Entrez deux nombres : 42 35
le plus petit diviseur de 42 et 35 est 7
```

## Exemple goto: calcul du plus petit commun diviseur

```
#include <stdio.h>
void main()
{
    int i = 0;
    while (i < 5)
    {
        printf("La variable i vaut %d\n", i);
        i++;
    }
}
```

=

```
#include <stdio.h>
void main()
{
    int i = 0;
    condition:
    if (i < 5)
    {
        printf("La variable i vaut %d\n", i);
        i++;
        goto condition;
    }
}
```

## Exemple `exit()`

- `exit()` normalement utilisée lorsqu'un programme veut se terminer à tout moment.
- La fonction `exit()` met fin à l'exécution du programme et renvoie le contrôle au système d'exploitation.
- La syntaxe de la fonction `exit ()` est,

`exit(status);`

Statut	Description
0	Le programme s'est terminé normalement.
1	Indique que le programme s'est terminé avec une sorte d'erreur. La valeur de retour est généralement ignorée.

# Exemple `exit()` : allocation dynamique

```
#include <stdio.h>
void main()
{
    int* memoireAllouee = NULL;

    memoireAllouee = malloc(sizeof(int));
    if (memoireAllouee == NULL) // Si l'allocation a échoué
    {
        exit(0); // On arrête immédiatement le programme
    }

    // On peut continuer le programme normalement sinon
}
```

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    if(argc <= 1) {
        printf("Parametres attendue !\n");
        exit(255);
    }
    else {
        printf("Processus normal...\n");
    }
    return 0;
}
```

## Fonction `atexit()`

- Utilisé pour spécifier ou enregistrer une ou plusieurs fonctions qui sont automatiquement exécutées à la fin du programme.
- Fonction `atexit()` de traitement de sortie qui s'exécute avant la fin du programme
- Lorsque toutes les fonctions enregistrées par `atexit()` sont exécutées, le programme se termine et renvoie le contrôle au système d'exploitation.
- Le prototype de la fonction `atexit()` se trouve dans le fichier `stdlib.h` et la syntaxe est,

```
int atexit (void (* funct) (void));
```

- où `funct` est la fonction à appeler.



## Fonction `system ()`

- La fonction `system ()` permet l'exécution de la commande du système (OS) à partir du programme en cours d'exécution C.
- Cela peut être très utile, par exemple, permettre au programme de lister un répertoire ou formater un disque sans quitter le programme.
- Doit inclure le fichier d'en-tête `stdlib.h`. La syntaxe est,

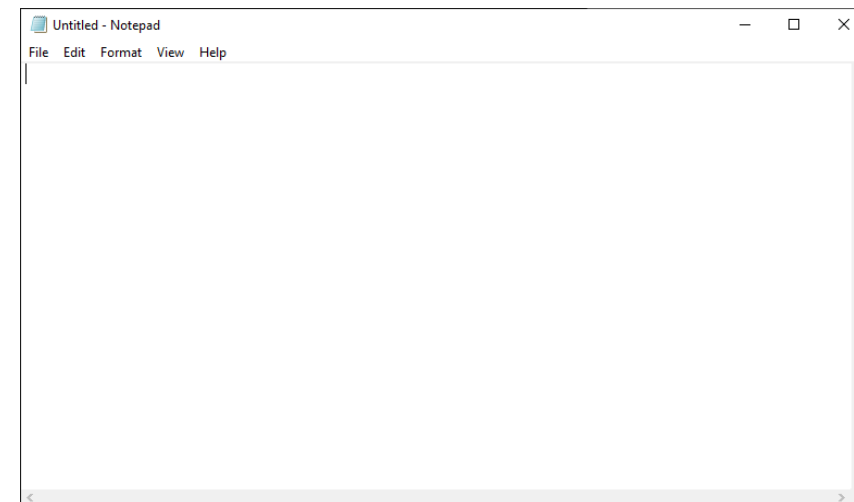
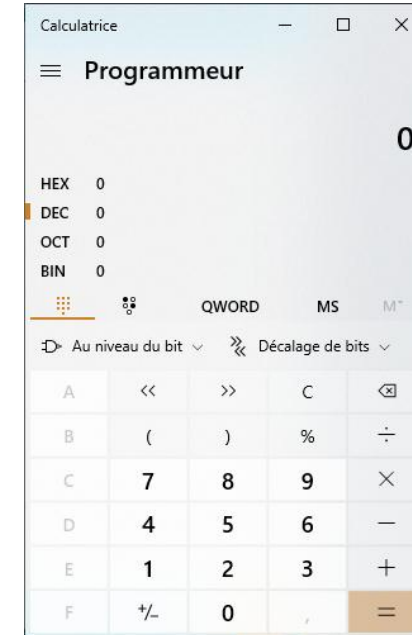
```
system("commande");
```

- La commande peut être une constante de chaîne ou un pointeur vers une chaîne.
- Une fois la commande OS exécutée, le programme continue à l'emplacement suivant immédiatement l'appel `system ()`.
- Si la commande passée à la fonction `system ()` n'est pas une commande de système d'exploitation valide, une commande incorrecte ou un message d'erreur de nom de fichier s'affiche avant de retourner au programme.
- La commande peut également être n'importe quel fichier exécutable ou batch à exécuter.

# Exemple system ()

```
#include <stdio.h>

void main()
{
    int iNum;
    system("calc");
    system("notepad");
    printf("\nOn lance deux programmes.\n");
}
```



## Mot clé **return**

- L'instruction **return** a une forme,

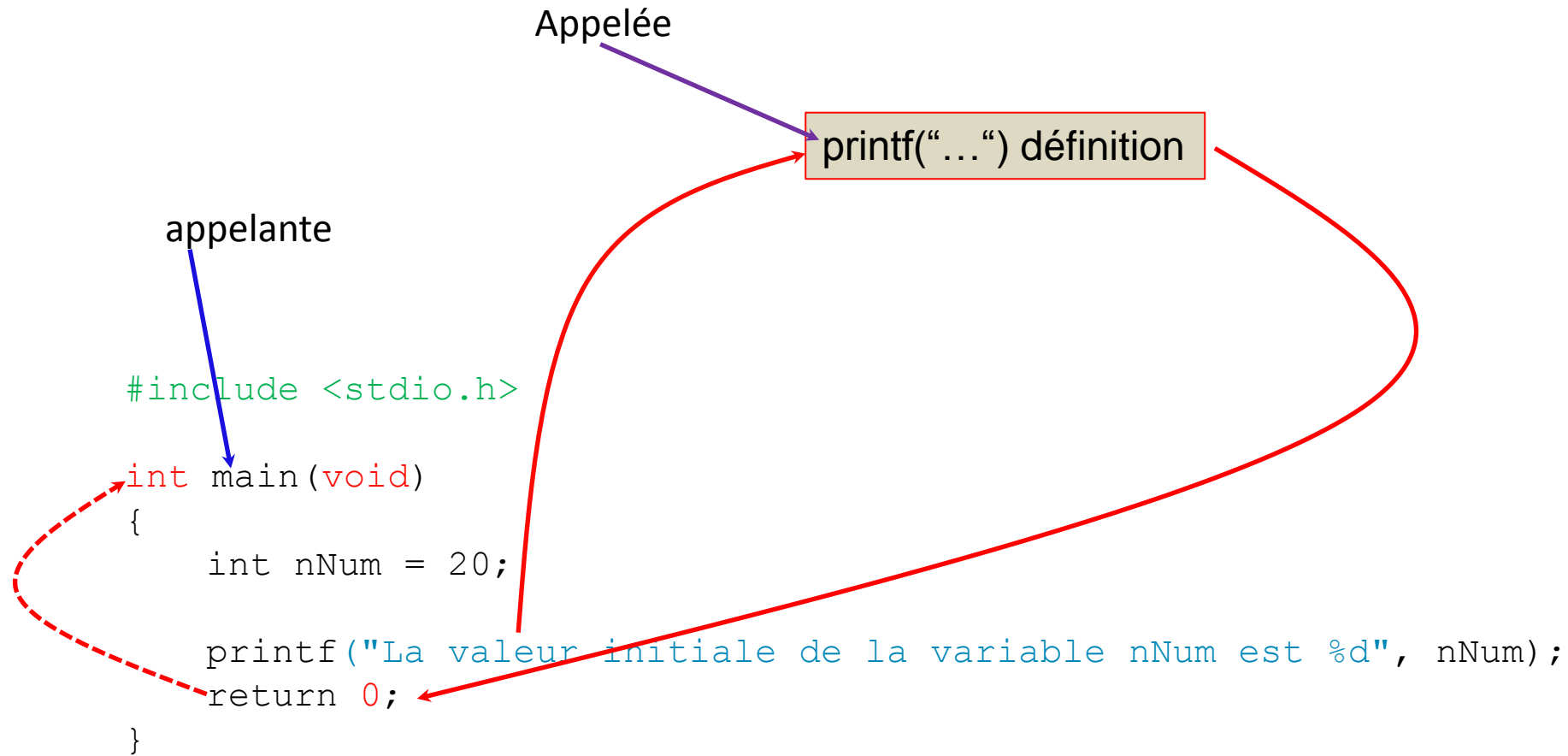
**return** *expression* ;

- L'action consiste à mettre fin à l'exécution de la fonction en cours et à transmettre la valeur contenue dans l'expression (le cas échéant) à la fonction qui l'a invoquée (citée).
- La valeur renvoyée doit être du même type ou *convertible au même type* que le type de retour de la fonction (conversion de type).
- Plusieurs instructions de retour peuvent être placées dans une fonction.
- L'exécution de la première instruction **return** dans la fonction met automatiquement fin à la fonction.

# Fonction main et return

- La fonction **main()** a un type par défaut **int** car elle renvoie la valeur **0** (un entier) à l'environnement.
- Une fonction de type **void** n'aura pas le mot-clé **return**. Au lieu de cela, dans ce cas, nous pouvons supprimer complètement l'instruction de **return**.
- Si une fonction appelle une autre fonction avant qu'elle ne soit définie, alors un *prototype doit être inclus dans la fonction appelante*.
- Cela donne des informations au compilateur pour rechercher la fonction appelée .

# Exercice d'exemple : 1



## Exercice d'exemple : 2

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

```
// prototype
```

```
void DisplayInteger(int);
```

```
void main(void)
```

```
{
```

```
int nNum = 30;
```

```
DisplayInteger(nNum);
```

```
}
```

```
void DisplayInteger(int iNum)
```

```
{
```

```
printf("The integer is %d\n", iNum);
```

```
}
```

appelante

Appelée

## Mot clé **break**

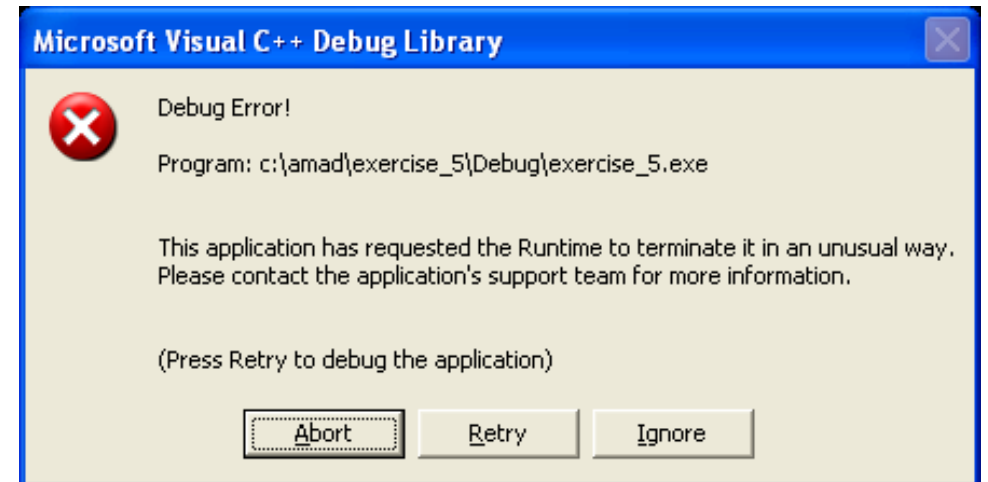
- Déjà discuté dans les constructions de **switch-case**.
- L'instruction **break** termine l'exécution de la boucle englobante ou de l'instruction conditionnelle la plus proche dans laquelle elle apparaît. Le contrôle passe à l'instruction qui suit l'instruction terminée.
- Il est utilisé avec l'instruction **switch** conditionnelle et avec les instructions de boucle **do**, **for** et **while**.
- Dans une instruction **switch**, **break** oblige le programme à exécuter l'instruction suivante après le basculement. Sans une instruction **break**, chaque instruction de l'étiquette de cas correspondant à la fin du commutateur, y compris la valeur par défaut, est exécutée.
- Dans les boucles, **break** termine l'exécution de l'instruction **do**, **for** ou **while** la plus proche. Le contrôle passe à l'instruction qui suit l'instruction terminée.
- Dans les instructions imbriquées, l'instruction **break** met fin uniquement à l'instruction **do**, **for**, **switch** ou **while** qui l'entoure immédiatement. Vous pouvez utiliser une instruction **return** ou **goto** pour transférer le contrôle à partir de structures plus profondément imbriquées.

# Fonctions `abort()` et `terminate()`

fonction	Description
<code>abort()</code>	Abandonne le processus en cours et renvoie le code d'erreur défini dans <code>stdlib.h</code>
<code>terminate()</code>	Utilisé lorsqu'un gestionnaire d'exception est introuvable. L'action par défaut pour terminer est d'appeler <code>abort ()</code> et provoque l'arrêt immédiat du programme. Il est défini dans <code>except.h</code> .

La syntaxe est : `void abort (void);`

1. `abort ()` ne renvoie pas le contrôle au processus appelant. Par défaut, il met fin au processus en cours et renvoie un code de sortie de 3.
2. Par défaut, la routine d'abandon imprime le message:





## Mot clé EOF

- Nous utilisons **EOF** (acronyme, signifie End Of File), a normalement la valeur  $-1$ .
- L'utilisateur tape une combinaison de touches dépendant du système pour signifier la fin du fichier, ce qui signifie «*Je n'ai plus de données à saisir*».
- **EOF** est une constante entière symbolique définie dans le fichier d'en-tête **<stdio.h>**.
- Les combinaisons de touches pour entrer **EOF** dépendent du système.
- Sur les systèmes UNIX et bien d'autres, l'**EOF** est <Touche Retour> ou **ctrl-z** ou **ctrl-d**.
- Sur un autre système tel que Microsoft Corp MS-DOS, l'**EOF** est **ctrl-z**.

```
Enter the letter grades.  
Enter the EOF character, ctrl-c or  
ctrl-z, etc to end input.  
ddddFFFaaaaEEEEccFFdecadf  
^Z  
  
Totals for each letter grade are:  
A: 5  
B: 0  
C: 3  
D: 6  
E: 5  
F: 6
```