**Langage de programmation**

**HAUTE ÉCOLE DE NAMUR-LIÈGE-LUXEMBOURG**

**Bloc 1**

Atelier 3 – Répétitives

Objectifs

* Distinguer les différentes formes de répétitives
* Définir la notion de portée d’une variable

[Introduction 1](#_Toc138414719)

[A. Qu’est-ce qu’une répétitive ? 2](#_Toc138414720)

[B. while… 3](#_Toc138414721)

[C. For 6](#_Toc138414722)

[D. Portée des variables 9](#_Toc138414723)

# Introduction

Dans cet atelier, vous allez apprendre à modifier l’ordre dans lequel les instructions sont exécutées, appelé le **flux des instructions**, au moyen d’instructions de **contrôle de flux**. Les instructions de contrôle de flux permettent, par exemple, de faire un choix entre deux actions, répéter une ou plusieurs actions…

La deuxième catégorie d’instructions de contrôle de flux est la répétitive. Le but de cet atelier est non seulement de vous permettre de comprendre les différentes façons de programmer une boucle en C, mais aussi de vous apprendre à les utiliser à bon escient.

Dans ce document, plusieurs conventions sont utilisées :

* les mots gras désignent des termes de vocabulaire liés à l’**informatique en général**.
* les mots soulignés et gras désignent des termes de vocabulaire directement liés aux cours de **programmation**.
* le logo signifie que vous avez quelque chose à réaliser.
* le logo est associé aux cadres présentant certaines conventions.
* le logo est associé aux cadres présentant les éléments liés à la propreté/lisibilité du code (*clean code*).

# Qu’est-ce qu’une répétitive ?

Une répétitive est une instruction de contrôle de flux, au même titre que les alternatives. Elle permet de **répéter** un certain nombre de fois (voire indéfiniment) une séquence d’instructions. Dans le langage commun, vous entendrez également parler de « boucles ».

Pour rappel, et parce que la confusion est encore fréquente dans le monde de la programmation, chacun des types de répétitive s'utilise dans des cas bien précis.

|  |  |  |
| --- | --- | --- |
| Itérative | | |
| Pseudo-code | Diagramme d'actions | Langage C |
| Tant que i 5  i i + 2  Fin Tant que | ╔══ while(i ≤ 5) ║ i = i + 2 ╙── | while (i <= 5) {  i = i + 2;  } |
| Répéter  i i - 1  Jusqu'à i 0 | ╔══ do ║ i-- ╙── until (i < 0) |  |
| Répéter  i i - 1  Tant que i 0 | ╔══ do ║ i-- ╙── while (i 0) | do {  i--;  } while (i >= 0); |
| Pour i allant de 1 à 10  k k + i  Fin Pour |  | for (i = 1 ; i <= 10 ; i++) {  k += i;  } |

Tableau 1 – comparaison (cours de Math)

Comme le tableau le montre, en C, il y a trois syntaxes différentes pour répéter un morceau de code. La première abordée est l’instruction while qui est la façon la plus générale de faire une boucle. La deuxième est l’instruction for, qui n’est utilisé que quand on connait le nombre d’itérations. Et la dernière est l’instruction do…while qui permet d’exécuter au moins une fois les instructions avant de poser la condition.

Chacune de ces instructions de contrôle de flux est abordée plus précisément dans la suite de cet atelier.

À nouveau, dans le cadre des répétitives, vous aurez besoin d’écrire des conditions. Veillez à bien réfléchir à celles-ci !

# while…

La forme la plus simple de répétitive est :

**TANT QUE** *condition* **FAIRE** *bloc d'instructions*

Cela signifie que, tant que l’évaluation de la condition donne la valeur true, le bloc d’instructions qui suit le **FAIRE** est exécuté. Dès que cette valeur devient false, ce sont les instructions situées juste après la répétitive qui sont exécutées.

Comme vous l’avez vu en « Principes de programmation », le diagramme d’actions correspondant est le suivant.

╔══ while (*condition*)

║ *instructions si la condition est vraie*  
╙──

En C, la répétitive « while » est composée d’une **instruction d’en-tête**, celle qui contient while suivi de la condition entre parenthèses ().

**while (**condition**)**

**{**

// bloc d’instructions si true

**}**

Comme vous l’avez vu en PP, il est possible de faire une boucle dont la condition porte sur une variable **initialisée** avant la boucle et **mise à jour** à chaque itération. Voici quelques instructions qui permettent d’afficher les chiffres de 1 à 7 (inclus).

int i = 1;

while (i <= 7) {

printf("%d ", i);

i++;

}

printf("Fin");

Pour rappel, toutes les instructions d’un même bloc doivent être indentées exactement au même niveau.

Écrivez ces instructions dans un nouveau fichier source compteJusque7.c, sans oublier d’ajouter les éléments nécessaires à son exécution.

Mais i, ce n’est pas un bon nom de variable !?

Le nom de variable i est parfois critiqué à juste titre, mais ce nom se trouve être approprié quand il s’agit d’une boucle constituée de **seulement quelques instructions** et dans laquelle le i ne sert qu’à passer à l’itération suivante. En effet, dans ce cas, la portée de la variable est suffisamment limitée pour ne pas induire de doute quant à sa signification. Pour information, le i vient des mathématiques, où la lettre **i** est utilisée comme **indice** du symbole de sommation ∑.

Exécutez-le ! Que produit-il ? Avant de lire les explications qui suivent, prenez le temps de décrire ce qui se passe de la manière la plus détaillée possible en FRANÇAIS.

Le mot while signifie « tant que » en anglais. Cette instruction indique qu’il lui faut répéter le bloc d’instructions qui suit tant que la condition est vraie. Dans ce cas-ci, on recommence tant que la valeur de i est inférieure ou égale à 7.

L’instruction while provoque la réalisation des étapes suivantes :

* la condition est évaluée.
* si la valeur qui en résulte est false, alors le bloc d’instructions qui suit est ignoré et l’exécution du programme se poursuit à la première instruction qui suit le bloc en question ; dans cet exemple printf("Fin").
* si la valeur qui en résulte est true, alors le bloc d’instructions aussi appelé le **corps de la boucle**, est exécuté :
  + l’appel de la fonction printf() pour afficher la valeur courante de la variable i suivie d’un espace.
  + l’instruction i++ qui incrémente/augmente de 1 le contenu de la variable i.

L’exécution de ces deux instructions consiste en la réalisation d’une première **itération**. Ensuite le programme boucle, c’est-à-dire que l’exécution reprend à la ligne contenant l’instruction while. La condition est à nouveau évaluée, et ainsi de suite tant que l’évaluation de cette condition donne true.

On parle donc aussi de structure **itérative**.

Boucle qui ne commence/finit jamais

La variable évaluée dans la condition doit avoir été initialisée au préalable. Si ce n’est pas le cas, l’évaluation de la condition donne un résultat indéfini ! Or, si la condition est évaluée à false dès le départ, le corps de la boucle n’est jamais exécuté. Cette boucle n'a donc pas de commencement. Il faut donc veiller à ce que la condition puisse être évaluée à true au moins une fois, sans quoi la boucle n’a aucun sens !

Testez le script suivant pour bien visualiser ce type d’erreur :

int i = 3;

while (i < 3) {

printf("Hello !");

i++;

}

Si la condition est toujours évaluée à true, le corps de la boucle est répété indéfiniment. C’est une boucle sans fin ou boucle infinie. Il faut donc veiller à ce que le corps de la boucle contienne au moins une instruction qui change la valeur d’une variable (il pourrait y en avoir plusieurs) intervenant dans la condition, de manière que cette condition soit évaluée à false et la boucle se termine.

Le script suivant est une boucle sans fin ! Testez-le pour bien visualiser ce type d’erreur :

int i = 1;

while (i < 3) {

printf("Hello !");

}

Vous pouvez interrompre l’exécution d’un programme via les touches CTRL + C.

Un autre cas de figure est celui dans lequel la condition de la boucle porte sur une variable dont la valeur est **obtenue au fur et à mesure des itérations**. Par exemple, imaginez que ce soit le mois des soldes sur le site de GOG (*Good Old Games*) et que vous vouliez faire un programme vous permettant de calculer rapidement la somme des achats effectués durant ce mois.

Pour déterminer ce montant total, il vous faut entrer les montants des jeux jusqu’à ce que vous ayez entré tous les montants, mais vous ne connaissez pas le nombre exact de jeux que vous avez achetés. Il faut donc que vous choisissiez une façon de dire « j’ai fini » au programme. Dans ce cas-ci, pour signifier que vous avez entré tous les montants, vous prévoyez de terminer par un montant de 0 euro. La condition de la boucle est donc « tant que le montant n’est pas de 0 euro ».

Pour pouvoir évaluer cette condition, il faut d’abord que la variable contenant le montant soit garnie avec la valeur du premier montant. Avant de rentrer dans la boucle, vous devez donc demander le premier montant. Ensuite, vous demandez le montant suivant à chaque itération. Lorsque tous les montants sont entrés, vous n’avez plus qu’à entrer 0 comme montant et la boucle se termine.

Le diagramme d’actions correspondant à ce programme est le suivant :

┌───\* MontantAchats  
│ montantTotal = 0  
│

│ sortir "Montant du jeu : "  
│ obtenir montantJeu  
│╔══ while (montantJeu ≠ 0)  
│║

│║ montantTotal += montantJeu  
│║

│║ sortir "Montant du jeu : "   
│║ obtenir montantJeu  
│╙──

│ sortir "Vous avez acheté pour un montant de ", montantTotal, " euros."

└──

Traduisez ce DA en programme C dans le fichier source achatsGOG.c. Exécutez-le !

Observez la différence entre ces deux façons de procéder (compteJusque7.c et achatsGOG.c). En effet, dans le premier cas, le programme n’a pas besoin de l'utilisateur pour s’arrêter, alors que dans le second, c’est lui qui décide quand la boucle se termine.

# For

En C, lorsque le nombre d’itérations est connu, on utilise l’instruction for.

La répétitive prend alors la forme suivante :

**POUR** *indice allant de … à …* **FAIRE** *bloc d'instructions*

Cela signifie que, tant que l’évaluation de la condition sur l’indice donne la valeur true, le bloc d’instructions qui suit le **FAIRE** est exécuté. Dès que cette évaluation donne false, ce sont les instructions situées juste après la répétitive qui sont exécutées.

Le diagramme d’actions correspondant est le suivant.

*initialisation\_indice*   
╔══ while (*condition\_sur\_indice*)  
║ *instructions si la condition est vraie*  
║ *mise\_à\_jour\_indice*  
╙──

En effet, l’instruction for est utilisée dès que la boucle porte sur une variable qui est **initialisée** avant la boucle et **mise à jour** à chaque itération, comme vous l’avez observé dans le programme compteJusque7.c.

Voici les instructions qui permettent de compter jusqu’à 7 en respectant la syntaxe du for.

for (int i = 1 ; i <= 7; i++) {

printf("%d ", i);

}

printf("Fin");

En C, la répétitive « for » est composée d’une **instruction d’en-tête**, celle qui contient for suivi de trois instructions entre parenthèses () : l’initialisation de l’indice, l’évaluation de la condition et la mise-à-jour de l’indice. Ces trois instructions doivent être séparer par des ;. En effet, ils permettent à l’analyseur syntaxique de les différencier au sein des ().

**for(***initialisation\_indice* **;** *condition\_sur\_indice***;** *mise\_à\_jour\_indice***)** **{**

// bloc d’instructions si true

**}**

À éviter absolument…

Les boucles « for » sont fréquemment utilisées de façon abusive. En effet en plus des éléments cités ci-dessus, il est possible de « compliquer » l’entête de cette répétitive.

Aucune explication n’est donnée à ce sujet parce que ce genre d’écriture, que vous risquez de rencontrer sur la toile, rend le code illisible voir incompréhensible !

Le mot for signifie « pour » en anglais. Cette instruction indique qu’il lui faut répéter le bloc d’instructions qui suit pour toutes les valeurs d’indice allant de la valeur initiale à la valeur précisée dans la condition. Dans ce cas-ci, on recommence pour toutes les valeurs de i allant de 1 à 7 inclus.

L’instruction for provoque la réalisation des étapes suivantes :

* l’indice est initialisé.
* la condition est évaluée.
* si la valeur qui en résulte est false, alors le bloc d’instructions qui suit est ignoré et l’exécution du programme se poursuit à la première instruction qui suit le bloc en question ; dans cet exemple printf("Fin").
* si la valeur qui en résulte est true, alors le bloc d’instructions aussi appelé le **corps de la boucle**, est exécuté : l’appel de la fonction printf() pour afficher la valeur courante de la variable i suivie d’un espace.
* l’instruction i++ qui incrémente/augmente de 1 le contenu de la variable i. Ensuite le programme boucle, c’est-à-dire que l’exécution reprend à l’évaluation de la condition, et ainsi de suite tant que l’évaluation de cette condition donne true.

Attention, si une autre condition que celle portant sur l’indice intervient, l’utilisation du for ne se justifie plus.

Dans le fichier source tableDe7.c, écrivez le programme qui permet d’afficher la table de multiplication (de 1 à 10) du chiffre 7 en affichant chaque calcul sur une ligne :

1 x 7 = 7  
2 x 7 = 14  
…

Sauvez ce programme dans tablesDeN.c et faites-en sorte que cette deuxième version affiche la table de multiplication (de 1 à 10) d’un nombre demandé et récupéré dans la variable nb.

Si le nombre nb vaut 5, l’affichage est :

Nombre : 5  
1 x 5 = 5  
2 x 5 = 10  
…

Dans les deux programmes qui suivent, on peut utiliser soit le while soit le for et ce de manière équivalente.

Écrivez le code suivant dans un nouveau fichier source somme.c. Exécutez-le et observez son fonctionnement.

int nombre;

int somme = 0;

for (int i = 1 ; i <= 5 ; i++) {

printf("Nombre : ");

scanf\_s("%d", &nombre);  
 somme += nombre;  
}

printf("Somme : %d", somme);

Transformez le for en while et exécutez-le à nouveau. Observez son fonctionnement. Assurez-vous que les résultats des deux versions soient identiques.

Écrivez le code ci-dessous dans un nouveau fichier source points.c. Exécutez-le et observez son fonctionnement. Quel est son but ?

int nbParties;

int sommePoints = 0;

nbParties = 0;  
while (nbParties < 10) {

int pointsPartie;

printf("Points de la partie : ");

scanf\_s("%d", &pointsPartie);  
 sommePoints += pointsPartie;  
 nbParties++;  
}

printf("Score moyen : %.2f", (double)sommePoints / nbParties);

Son but est le suivant : en supposant que le joueur a joué 10 parties et qu’il n’a obtenu aucun point négatif ou nul à une partie, voici un programme qui permet de calculer la moyenne de ses points sur les 10 parties et de l’afficher.

Transformez le while en for et exécutez-le à nouveau. Observez son fonctionnement. Assurez-vous que les résultats des deux versions soient identiques !

Il se peut que vous rencontriez un problème lié à l’endroit où vous avez déclaré la variable nbParties. Réfléchissez et essayez de trouver la solution. Si vous ne trouvez pas, référez-vous au point E de cet atelier sur la portée des variables.

Reprenez le programme points.c mais, cette fois, on veut permettre à l’utilisateur de quitter la boucle avant d’avoir encoder les points des 10 parties (parce qu’il en a joué moins), et ce en entrant 0 comme points. Il ne peut toujours pas avoir de points négatifs ou nuls.

Commencez par réfléchir à ce que vous allez devoir ajouter et à l’endroit où vous allez l’ajouter.

Quelle version de boucle allez-vous choisir, la version avec un while ou celle avec un for ?

Réfléchissez bien ! Quelle est la « condition » pour utiliser un for ? La réponse se trouve dans la première ligne de ce point !

Pensez-vous que cette « condition » est respectée lorsqu’on permet à l’utilisateur de quitter ainsi la boucle quand il le désire ?

N’hésitez pas à montrer vos solutions à d’autres étudiants afin d’en discuter…

# Portée des variables

Dans la série d’exercices introductifs (Exercice 0), vous avez découvert ce qu’est une variable et ses caractéristiques. Parmi celles-ci se trouve la portée…

En effet, une variable n’est pas nécessairement accessible partout dans le programme. De plus, les variables ne sont pas toutes accessibles en même temps et pour la même durée. La partie du programme dans laquelle la variable est accessible est appelée sa **portée**, et la durée pendant laquelle la variable existe sa **durée de vie**.

Sa portée et sa durée de vie dépendent donc de l’endroit où la variable est définie.

Copiez le code du programme points.c dans portée.c. Dans le cas du while vous ne devriez pas avoir rencontré de problèmes, mais dans le cas du for, si vous l’avez écrit comme proposé ci-dessous, vous devriez avoir eu un problème à la compilation.   
En effet, compilez le bout de code suivant.

void main(void) {

// version 1

int sommePoints = 0;  
  
 for (int nbParties = 0; nbParties < 10 ; nbParties++) {

int pointsPartie;

printf("Points de la partie : ");

scanf\_s("%d", &pointsPartie);  
 sommePoints += pointsPartie;  
 }

printf("Score moyen : %.2f", (double)sommePoints / nbParties);

}

Quel est le message d’erreur affiché ? Notez l’instruction qui provoque cette erreur : c’est lors de l’affichage du score moyen que l’erreur se produit !

À votre avis, pourquoi ce message d’erreur est-il généré et pourquoi lors de cette instruction précisément ?

Modifiez le code du fichier source portée.c de manière à éliminer cette erreur. Le code doit donc être le suivant (les modifications sont mises en évidence via le soulignement ondulé).

void main(void) {

// version 2

int sommePoints = 0;  
 int nbParties;

for (nbParties = 0; nbParties < 10 ; nbParties++) {

int pointsPartie;

printf("Points de la partie : ");

scanf\_s("%d", &pointsPartie);  
 sommePoints += pointsPartie;  
 }

printf("Score moyen : %.2f", (double)sommePoints / nbParties);

}

Compilez à nouveau ce programme. Y’a-t-il encore un message d’erreur ?

Avez-vous compris pourquoi l’erreur n’est plus levée ?

En fait, il s’agit d’un problème lié à la portée de la variable et donc à l’endroit où elle est déclarée. En effet, dans la « version 1 », la déclaration se trouvant dans la partie initialisation du for, elle n’est accessible qu’à l’intérieur de ce bloc (entre les accolades). Lors de son utilisation dans l’instruction d’affichage du score moyen, elle n’est pas définie et donc pas accessible.

Dans la « version 2 », la déclaration a été déplacée au niveau du bloc de la fonction principale. La variable est alors accessible dans le bloc qui constitue cette fonction et donc dans l’instruction d’affichage du score moyen.

En C, la norme impose que les variables soient déclarées en début de bloc. De plus, il est d’usage de ne les déclarer que dans le bloc où on en a besoin. La « version 3 » proposée ci-dessous respecte la norme ainsi que cette convention en déclarant la variable pointsPartie seulement dans le bloc où elle est utilisée, c’est-à-dire dans le for.

void main(void) {

// version 3

int sommePoints = 0;  
 int nbParties;

for (nbParties = 0; nbParties < 10 ; nbParties++) {

int pointsPartie;

printf("Points de la partie : ");

scanf\_s("%d", &pointsPartie);  
 sommePoints += pointsPartie;  
 }

printf("Score moyen : %.2f", (double)sommePoints / nbParties);

}

D’autres niveaux existent, mais ils seront abordés plus en détail dans le module sur les fonctions.