**Langage de programmation**

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

**Bloc 1**

Atelier 1 – Instructions, expressions et opérateurs

Objectifs

* Distinguer les instructions et les expressions
* Identifier un opérateur et l’utiliser
* Approfondir certains éléments de base : variable, type, littéral, affichage et lecture

[Introduction 1](#_Toc115864758)

[A. Particularités des printf et scanf 3](#_Toc115864759)

[B. Instructions et expressions 5](#_Toc115864760)

[C. Opérateurs 6](#_Toc115864761)

[D. Opérateurs particuliers 13](#_Toc115864762)

# Introduction

Pour les distraits ou les absents, voici à nouveau quelques mots de présentation : le cours « Langage de programmation » fait partie de l’Unité d’Enseignement « Programmation » et est donné en parallèle avec le cours de « Principes de programmation ». Il s’agit d’un cours d’introduction à la programmation au moyen d’un langage procédural structuré : le « C ».

Afin de vous permettre de découvrir chaque élément théorique petit à petit, les cours sont composés de trois types d’activités :

* des **ateliers**, permettant de présenter brièvement certains éléments théoriques/syntaxiques au travers de petits exercices dirigés. N’hésitez pas à le faire à plusieurs…
* quelques **mises en commun**,permettant de revoir ensemble les concepts et éléments de syntaxe vus lors des ateliers. Elles sont données en grands groupes.
* des **exercices** moins dirigés. Ils seront en partie réalisés lors de séances d’exercices, en petit groupe. Il est important d’y participer de façon active !

Les ateliers sont à réaliser en préparation des mises en commun. **Il est donc important de prendre connaissance du planning et de le respecter au mieux** !

Pour pouvoir constater le bon fonctionnement de vos programmes, il vous est demandé d’utiliser le compilateur en ligne de commande associé à Visual Studio comme présenté lors des deux premières séances.

Comme vous le savez déjà, vous n’aurez pas le temps de faire tous les exercices lors des séances, il est donc nécessaire de vous assurer d’avoir compris la matière en travaillant en dehors de ces séances !

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*).

Ce premier atelier est à réaliser en préparation du deuxième cours de théorie.

Vous allez d’abord approfondir la notion de format d’affichage et de lecture.

Ensuite vous allez revoir la notion d’instruction et découvrir celle d’expression.

Les deux points suivants concernent les opérateurs qui peuvent être utilisés dans les expressions et donc dans les instructions.

# Particularités des printf et scanf

Il est important de bien choisir les formats de lecture et d’affichage.

Par exemple, dans le cas du printf, à votre avis (sans exécuter les instructions), que va afficher le bout de code ci-dessous ?

int nbEntier = 25;

double nbReel = 25.5;

printf("Entier : %f\n", nbEntier);

printf("Reel : %d\n", nbReel);

// Une erreur ?... Car il faut inverser les deux %f et %d : %f est pour les nombres à virgules float et double et %d est pour les entier.

Créez un nouveau fichier source particularites.c. Écrivez-y la fonction principale et ajoutez-lui ce bout de code. Compilez-le fichier source.

La compilation génère des messages d’avertissement (*warning*). Lisez-les et essayez de les comprendre !

Exécutez le programme malgré tout. L’affichage produit correspond-il à vos attentes ?

Recommencez en remplaçant les deux instructions d’affichage par les suivantes.

printf("Entier : %d\n ", nbEntier);

printf("Reel : %f\n ", nbReel);

Observez le résultat de la compilation : il n’y a plus de message d’avertissement ! Pourquoi ?

Ce phénomène est lié à l’encodage des nombres en mémoire, voir cours d’« Architecture des systèmes ».

Le même phénomène est observable dans le cas du scanf\_s.

Dans le programme particularites.c, après avoir choisi les bons formats d’affichage, modifiez le code source de manière à demander et récupérer les valeurs au lieu d’initialiser les variables.

Il faut, tout en gardant les déclarations de variables, éliminer les affectations et ajoutez les instructions ci-dessous (au bon endroit).

printf("Nombre entier : ") ;

scanf\_s("%lf", &nbEntier);

printf("Nombre reel (avec un .) : ") ;

scanf\_s("%d", &nbReel);

À votre avis qu’affiche le programme si les valeurs encodées sont respectivement 7 et 8,5 ?

Compilez le fichier source et observez le résultat de la compilation : il y a de nouveau des messages d’avertissement.

Exécutez le programme malgré tout. L’affichage produit correspond-il à vos attentes ?

Essayez de comprendre pourquoi le résultat ne convient pas et de modifier le code source de manière à le corriger.

Lorsque vous rencontrez un problème d’affichage de ce genre, n’oubliez pas de vérifier la cohérence entre le type de ce que vous affichez et le format d’affichage utilisé !

Toujours dans ce même fichier source particularites.c, ajoutez les instructions suivantes et exécutez le programme de manière à voir ce qui est affiché.

printf("Mois\tNombre de jours\n");

printf("%s\t%d\n", "Octobre", 31);

printf("%s\t%d\n", "Novembre", 30);

printf("%s\t%d\n", "Decembre", 31);

Que signifie selon vous les caractères '\t' et '\n' dans le format d’affichage ?

Ce sont des associations de caractères qui permettent d’afficher des caractères spéciaux comme la tabulation, le passage à la ligne et d’autres. Le '\' est ce qu’on appelle un caractère d’échappement. Il déclenche une interprétation différente du ou des caractères qui le suivent.

Compilez le fichier source et exécutez le programme. Vous remarquez que, dans ce cas-ci, la tabulation ne suffit pas à aligner les nombres de jours les uns avec les autres. Comment faire pour avoir tous les nombres alignés ?

Une solution est de faire en sorte que tous les libellés de mois occupent la même quantité de caractères lors de leur affichage. Pour obtenir ce résultat, ajouter un nombre suffisamment grand entre le '%' et le 's' du format d’affichage d’une chaîne de caractères, ce qui donne par exemple %10s. Recompilez le fichier source, exécutez le programme et observez le résultat.

Les mois sont maintenant tous alignés à droite. Ajoutez un '-' devant le nombre que vous avez choisi.

Recompilez le fichier source, exécutez le programme et observez le résultat.

Enfin, ajoutez l’instruction permettant d’afficher la phrase suivante : « Les travailleurs belges ont en moyenne 35 jours de congés par an, ce qui fait 9.59 % de l’année. » sans les accents…

Compilez une fois de plus le fichier source et exécutez le programme. Plusieurs messages d’avertissement apparaissent. Que signifient-il ?

Relisez la phrase et ciblez ce qui pose problèmes. Ajoutez le caractère d’échappement au bon endroit et essayez à nouveau.

Dans le cas du %, le caractère d’échappement est le '%' ! Recommencez une dernière fois et assurez-vous que le résultat est celui que vous espérez !

# Instructions et expressions

Ces deux termes représentent des notions distinctes, bien que liées l’une à l’autre.

## Instruction

Une **instruction** est un bout de code qui correspond à une **action**.

Il y a plusieurs formes d’instructions : la déclaration d’une variable, une affectation, un appel de fonction… Vous avez déjà vu la syntaxe de ces instructions lors des séances d’exercices précédentes.

Pour rappel, une variable porte un **nom** et est destinée à mémoriser une **valeur** dans une zone mémoire repérée par son **adresse**. C’est lors de sa **déclaration** que vous allez lui associer un **type**. Vous pouvez lui donner une valeur grâce à une **affectation**.

Tout cela peut se résumer en une ligne, comme par exemple

int reponseUniverselle = 42;

Créez un nouveau fichier source somme.c. Écrivez-y la fonction principale et ajoutez-lui le code source proposé ci-dessous.

int somme;

printf("somme vaut %d\n", somme);

somme = 2 + 3;

Compilez ce fichier source et exécutez le programme… Que constatez-vous ?

Sans modifier les instructions, remettez de l’ordre dans le programme afin qu’il affiche le message suivant :

Somme vaut 5

Comme vous l’avez compris, un programme est une suite d’instructions. Les instructions sont exécutées les unes à la suite des autres et forment ce qu’on appelle une **séquence d’instructions**.

L’ordre des instructions est primordial lors de l’exécution de vos programmes. Par exemple, une instruction affichant la valeur d’une variable, si elle est exécutée avant d’avoir affecté une valeur à celle-ci, n’affichera pas la valeur espérée, mais celle qui est à ce moment-là dans la zone mémoire correspondante !

Rappelez-vous qu’il possible de regrouper plusieurs instructions en **blocs d’instructions**.

## Expression

Dans une instruction, une valeur peut être un **littéral** (42, 'Z', 3.1415) mais elle peut aussi être le résultat de l’**évaluation d’une expression**.

Une **expression** est une combinaison de littéraux, de variables, d'opérateurs, etc. qui, lorsqu’on la calcule/l’évalue, produit une nouvelle **valeur** :

5 \* 8 nombreLu + 3 pi = 3.1415 lettre == 'a'

Il existe plusieurs opérateurs permettant de former des expressions. Le point suivant permet d’en aborder quelques-uns.

# Opérateurs

Les notions de variable et de littéraux ont déjà été abordées. Il vous reste donc à comprendre ce qu’est un opérateur et comment les utiliser.

Les **opérateurs** que vous connaissez pour le moment sont les opérateurs arithmétiques (+, -, \*, /, %) et ceux de comparaison (<, ≤, ≥, >, =, ≠). À ceux-ci s’ajoute l’opérateur d’affectation qui a déjà été abordé ci-avant.

## Opérateur d’affectation

En C, l’affectation se fait au moyen de l’opérateur =. L’affectation peut être à la fois considérée comme une expression et comme une instruction.

Il est à noter que d'autres langages considèrent l’affectation comme une instruction et ne permettent pas son utilisation dans une expression.

Quelques informations complémentaires peuvent être apportées à la notion d’affectation.

Modifiez le fichier source somme.c en remplaçant l’instruction

somme = 2 + 3;

par

2 + 3 = somme;

Compilez ce code source et observez ce qui est produit.

Vous devriez avoir un message d’erreur semblable à celui de la Figure 1.



Figure - L-Value

L’opérateur d’affectation = suit une règle bien précise et générale à la plupart des langages de programmation : c’est la valeur située à la droite du =, appelée **expression de droite**, qui est mémorisée dans la zone mémoire située à la gauche du =, appelée **expression de gauche**.

Le message d’erreur ci-dessus précise donc que l’opérande à gauche de l’opérateur = doit être une expression de gauche, c’est-à-dire une zone mémoire !

Corrigez le code source pour que l’instruction d’affectation soit correctement écrite. Recompilez le code source et vérifiez que tout se passe comme vous l’espérez.

On peut donc préciser la syntaxe de l’instruction d’affectation comme ceci :

<l-value> **=** <r-value>;

Il faut y remplacer

* <l-value> par une expression de gauche, une zone mémoire,
* <r-value> par une expression de droite, une valeur.

Il est important de noter que les expressions de gauche peuvent être utilisées en tant qu’expressions de droite. En effet comme une expression de gauche est une zone mémoire contenant une valeur, elle peut être utilisée dans une expression de droite.

Par exemple, une variable est à la fois

* une expression de gauche lorsqu’on la considère comme étant la zone mémoire dans laquelle on désire mémoriser une valeur,

int nbTorchesSac = 42;

* une expression de droite lorsqu’on la considère comme étant la valeur que l’on désire utiliser comme opérande d’une expression.  
  int nbTorchesAFaire = 64 - nbTorchesSac;

Reprenez le code source du fichier torches.c créé lors des séances d’exercices précédentes. Intégrez-y convenablement les instructions suivantes.

int nbTorchesMax = 64;

int nbTorchesSac = 42;

int nbTorchesAFaire;

nbTorchesAFaire = nbTorchesMax - nbTorchesSac;

Dans ce même fichier source, affichez les résultats de ces instructions, en passant

* soit par l’expression, comme vous l’avez déjà fait précédemment :

printf("Somme : %d", 2 + 3);

printf("Vous devez refaire %d torches.", nbTorchesMax - nbTorchesSac);

* soit par une variable, comme montré ci-dessous :

printf("Somme : %d", somme);

printf("Vous devez refaire %d torches.", nbTorchesAFaire);

Ces instructions d’affichage font intervenir une expression de droite, c’est-à-dire une valeur, soit sous la forme d’une expression, soit sous la forme d’une variable.

## Opérateurs arithmétiques

Petite précision sur l’opérateur - car il peut être envisagé de deux façons :

* s'il porte sur un seul opérande, on parle d’**opérateur unaire** : -x -(8)

*Remarque : Dans -8 et -5, le - fait partie de l’écriture du littéral.*

* s’il est utilisé entre deux opérandes, on parle d’**opérateur binaire** : x - y

Les autres opérateurs, listés plus haut, sont des opérateurs binaires puisqu’ils font intervenir deux opérandes : 5 != 8 3 \* 9 7 % 2

Créez un nouveau fichier source vacances.c dans lequel vous allez recopier le code source proposé ci-dessous. N’oubliez pas la fonction principale et les éventuelles bibliothèques. Complétez les instructions d’affichage en remplaçant chaque … par ce qui convient pour pourvoir afficher les valeurs demandées ! Dans les guillemets " ", il faut ajouter un format d’affichage et après la virgule , vous devez ajouter une expression.

int nbJoursVacances = 27;

int nbJoursVacancesEnsoleilles = 11;

double prixVacancesParJour = 7.50;

double budgetVacances = 300.00;

double depensesParJourEnsoleilleEnVacances = 8.25;

printf("Nombre de jours de vacances non ensoleilles : …\n", …);

printf("Prix total des vacances : …\n", …);

printf("Nombre de jours de vacances autorises par le budget : …\n", …);

printf("Prix correspondant aux jours de vacances ensoleilles : …\n", …);

printf("Depenses totales pour les jours de vacances ensoleilles : …\n", …);

Le résultat de l’affichage est le suivant : Une image contenant texte

Description générée automatiquement

Ajoutez les instructions ci-dessous. Complétez-les avec les expressions nécessaires à l’affichage des valeurs demandées ainsi que le format d’affichage adéquat.

printf("Nombre de semaines entieres passees en vacances : …\n", …);

printf("Nombre de jours de vacances en plus des semaines entieres : …\n", …);

Avez-vous trouvé la façon la plus efficace de répondre à ces deux dernières demandes ?   
Un opérateur que certains ne connaissent pas et qui est pourtant très utile est %, le modulo. Il permet de récupérer le reste de la division entière d’un nombre par un autre.

Dans la division euclidienne, que vous devriez avoir vu en primaire, voici les différents éléments de la division :

dividende

diviseur

23 4

quotient

- 20 5

reste

3

Dans le cas du /, si les deux opérandes sont entiers, on récupère le quotient, alors que dans le cas du %, on récupère le reste.

Le résultat de l’affichage est le suivant :

Une image contenant texte, orange

Description générée automatiquement

Afin de vous assurer que vous avez bien compris quand et comment utiliser les opérateurs / et %, ajoutez les déclarations et initialisations ci-dessous (au bon endroit) dans le code source de vacances.c, ainsi que les instructions proposées.

Déclaration de variable

En C, la norme précise que les déclarations doivent être placées en début de bloc. Pour rappel, un **bloc d’instructions** est une suite d’instructions délimitée par des accolades {…}.

Complétez à nouveau ces instructions comme précédemment.

int heureDepartHHMM = 1427;

int dureeVoyageEnMinutes = 134;

printf("Heure de l'heure de depart : …\n", …);

printf("Minutes de l'heure de depart : …\n", …);

printf("Temps de trajet : … h et … min\n", …);

printf("Nombre d'heures entieres dans la duree du voyage : …\n", …);

printf("Nombre de minutes de voyage en plus des heures entieres : …\n", …);

printf("Heure de l'heure d'arrivee : …\n", …);

printf("Minutes de l'heure d'arrivee : …\n", …);

Le résultat de l’affichage est le suivant : Une image contenant texte

Description générée automatiquement

Recommencez cet exercice, mais avec une heure de départ initialisée à 1143 et une durée de voyage en minutes initialisée à 92. N’oubliez pas de vérifier que votre réponse est plausible : les minutes ne peuvent pas excéder 59 !

Le résultat de l’affichage est le suivant :

Une image contenant texte

Description générée automatiquement

## Opérateurs de comparaison

Les opérateurs de comparaison servent à comparer des valeurs, et ont pour résultat une valeur booléenne.

Deux précisions doivent être apportées concernant la syntaxe des opérateurs de comparaison :

* dans le cadre de la rédaction du code source d’un programme, vous ne pouvez pas utiliser les symboles ≤, ≥ et ≠. Ceux-ci sont respectivement replacés par <=, >= et != (le « ! » signifiant « *not* »/« pas », le symbole « != » signifiant ainsi « différent » ou « *not equals* »/« pas égal »).
* l’opérateur de comparaison = ne peut être écrit tel quel car il pourrait être confondu avec l’opérateur d’affectation. Or, le compilateur n’a pas à faire de choix quant à la signification d’un opérateur… La syntaxe de l’opérateur d’égalité est donc ==.

Toujours dans ce même fichier vacances.c, ajoutez et complétez les instructions ci-dessous.

printf("Il n'a jamais fait ensoleille durant les vacances : %s\n", … ? "vrai" : "faux");

printf("Les vacances ont dure moins de 15 jours : %s\n",… ? "vrai" : "faux");

printf("Il a fait ensoleille tous les jours des vacances : %s\n", … ? "vrai" : "faux");

printf("Le budget n'est pas suffisant pour couvrir le prix des jours de vacances : %s\n", … ? "vrai" : "faux");

printf("Le voyage dure moins de 2 heures : %s\n", … ? "vrai" : "faux");

printf("Le depart est prevu apres midi : %s\n", … ? "vrai" : "faux");

printf("Le cout total des vacances (prix par jour + depenses des jours ensoleilles) depasse le budget : %s\n", … ? "vrai" : "faux");

printf("Le cout moyen d'un jour de vacances n'atteint pas les 12 euros : %s\n", … ? "vrai" : "faux");

L’opérateur conditionnel ? : est approfondi dans la partie sur les opérateurs particuliers.

Le résultat de l’affichage est le suivant :

Une image contenant texte

Description générée automatiquement

Pour ceux qui ont des difficultés à répondre à cette question, commencez par demander des explications à d’autres étudiants… discutez-en !

Pour ceux qui sont complètement « bloqués », la solution pour répondre à la première proposition est nbJoursVacancesEnsoleilles == 0. À vous de trouver les autres solutions…

## Priorités des opérateurs

Les expressions peuvent être plus complexes en associant différents opérateurs. Afin de pouvoir évaluer ces expressions, il est nécessaire de donner des priorités aux opérateurs.

Vous connaissez probablement déjà certaines de ces priorités comme celle concernant les opérateurs + et \*. En effet, dans l’expression 3 + 4 \* 5, vous devez d'abord multiplier 4 par 5, et ensuite ajouter ce résultat à 3, obtenant ainsi le résultat de 23.

Ce mécanisme qui amène certains opérateurs à être pris en compte avant d'autres s’appelle la **priorité des opérateurs**/**opérations**.

Comme dans tout langage, la norme du C définit précisément les priorités de chacun des opérateurs et suppose que les opérateurs d'une priorité plus grande (plus élevée) sont effectués avant les opérateurs d'une priorité inférieure.

Dans l’expression ci-dessus, le \* a une priorité plus élevée que le +.

Évaluez les expressions suivantes sans passer par un programme. Faites-le également en ajoutant des parenthèses autour de la première sous-expression puis autour de la seconde sous-expression. Dans l’exemple ci-dessus, vous auriez eu les trois expressions suivantes : 3 + 4 \* 5, (3 + 4) \* 5 et 3 + (4 \* 5).

15 + 3 \* 2

3 \* 4 % 5

15 % 4 / 3

Créez un fichier source prioritesOperateurs.c et écrivez-y les instructions permettant d’afficher la valeur de ces expressions. Avez-vous obtenu les mêmes résultats ? Si non, à votre avis, quelle en est la raison ?

Voici le tableau des priorités des opérateurs arithmétiques :

Priorité

|  |  |
| --- | --- |
| -(unaire) | ← |
| \* / % | → |
| + -(binaire) | → |
| < <= > >= | → |
| == != | → |

c

Vous remarquez que certains opérateurs ont la même priorité, ils sont sur la même ligne. Cependant, il y a également un ordre à respecter dans l’évaluation d’expressions composées d’opérateurs ayant la même priorité. Dans le cas d’une expression telle que   
5 + 3 - 2, l’évaluation des expressions se fait de gauche à droite (→) 5 + 3, qui donne 8, pour ensuite évaluer 8 - 2. Pour d’autres opérateurs, l’évaluation aura lieu de la droite vers la gauche (←).

Sans les exécuter, donnez la valeur des variables et précisez le type du résultat après les instructions suivantes :

val1 = 10 + 6 / 2;

val2 = 3 \* 2 + 4 \* 3;

val3 = 2 <= 1 + 4;

val4 = 2 + 1 != 3;

val5 = 24 + 36 % 12;

Ajoutez ensuite ces instructions au code source prioritesOperateurs.c. Déclarez chaque variable en utilisant le bon type et affichez ces variables. Vérifiez vos réponses.

Cette notion semble futile dans certains cas, mais vous avez pu voir que cela peut engendrer des erreurs inattendues… pour ceux qui n’ont pas pris la peine de bien comprendre ces propriétés.

Dans les instructions suivantes, ajoutez des parenthèses uniquement là où elles sont absolument nécessaires pour que la valeur de l’expression soit celle de la valeur attendue.

printf("Expression : %d - Attendue : 8\n ", 10 + 6 / 2);

printf("Expression : %d - Attendue : 54\n ", 3 \* 2 + 4 \* 3);

printf("Expression : %d - Attendue : 0\n ", 24 + 36 % 12);

N’hésitez pas à les tester en les ajoutant au code source précédant.

Étant donné que le = est un opérateur, il a sa place dans le tableau des priorités. Et comme il est à effectuer en dernier lieu, il se situe à la fin de celui-ci.

|  |  |
| --- | --- |
| -(unaire) | ← |
| \* / % | → |
| + -(binaire) | → |
| < <= > >= | → |
| == != | → |
| = | ← |

Enlevez les parenthèses des expressions suivantes lorsqu'elles peuvent être retirées. Sur base des déclarations, vérifiez que la valeur de l’expression correspond toujours à la valeur attendue.

int a = 3;

int b = 7;

printf("Expression : %d - Attendue : 38\n ", (a \* 12) + (b - 5));

printf("Expression : %d - Attendue : 65\n ", (5 \* a) + 2 \* ((3 \* b) + 4));

printf("Expression : %d - Attendue : 825\n ", (5 \* (a + 2) \* 3) \* (b + 4));

## Vérification des types

Comme vous l’avez vu lors du point sur les formats d’affichage et de lecture, le type (de la valeur résultant de l’évaluation) d’une variable a des répercussions sur la façon dont elle peut être utilisée.

En effet, les valeurs entières ne peuvent être affichées (via un printf) avec un "%f", de même que pour les valeurs réelles et le "%d".

Le compilateur C a parmi ses nombreux rôles celui de vérifier que les types utilisés dans les instructions sont compatibles entre eux.

Avant d’utiliser le résultat d’une expression, il faut en déterminer le type afin de s’assurer que le résultat de l’instruction corresponde à vos attentes.

Ajoutez les instructions ci-dessous dans le fichier source prioritesOperateurs.c. Complétez-les avec le bon format d’affichage.

printf("…\n ", 800 \* 600);

printf("…\n ", 19410909 / 10000);

printf("…\n ", 19410909 / 100 % 100);

printf("…\n ", 19410909 % 100);

printf("…\n ", 25.7 + 24.8 + 32.4);

printf("…\n ", (25.7 + 24.8 + 32.4) / 3);

# Opérateurs particuliers

Il existe d’autres opérateurs un peu particuliers…

## Conversion de type ou transtypage

Dans le fichier source vacances.c, ajoutez l’instruction suivante et complétez-la de manière à afficher la valeur demandée.

printf("Pourcentage de jours ensoleilles sur les vacances : %f\n", …);

Dans ce cas, le résultat obtenu n’est pas correct et ce n’est pas qu’une question d’affichage. En effet, dans le cas d’une division d’un entier par un autre entier, le résultat attendu peut être un entier, comme pour le calcul de l’année ci-dessus. Mais, dans d’autres cas, le résultat de l’évaluation de l’expression doit être réel, comme pour un pourcentage.

Un autre exemple est le calcul d’une moyenne : on veut déterminer la note d’une unité d’enseignement (UE) sur base des notes des activités d’apprentissage (AA) de cette UE. Les AA sont évaluées sur 20 avec des notes entières. La note finale est, dans cet exemple, la moyenne des notes des AA et est arrondie à l’unité supérieure lorsque la première décimale atteint ou dépasse 0,5.

Si le calcul de la moyenne des AA se fait uniquement en entier, quel est, selon vous, le résultat de l’expression (15 + 13 + 10) / 3 ?

En C, la valeur de cette expression est entière !

Dans le cas de ce calcul de moyenne, le résultat de l’évaluation doit être réel ! Il faut qu’un des opérandes de la division soit un réel pour que l’évaluation donne un résultat réel. Pour cela, vu qu’il s’agit de littéraux, il suffit de mettre un de ceux-ci au format réel et le problème est réglé : (15 + 13 + 10) / 3.0 ou (15 + 13 + 10) / 3.. La valeur de l’expression est maintenant réelle.

Pour bien visualiser cette nuance, créez un fichier source moyenneUE.c, écrivez-y la fonction principale et l’inclusion de la bibliothèque. Ajoutez enfin les deux instructions suivantes :

printf("Moyenne des cotes de l'UE : %d\n", (15 + 13 + 10) / 3);

printf("Moyenne des cotes de l'UE : %f\n", (15 + 13 + 10) / 3.);

Bref, il est parfois intéressant de faire en sorte que l’évaluation d’une expression donne un résultat d’un type différent que celui produit par défaut. Selon les explications ci-dessus, la moyenne obtenue en réel doit être arrondie à l’unité supérieur. Il est donc intéressant de savoir que, si le calcul se fait en entier, vous perdez un point à votre UE !

Cela ne changerait pas le cours de votre vie, mais imaginez la même erreur d’arrondi dans le calcul de la trajectoire d’un bras articulé permettant des opérations à distance… (voir <https://www.lecho.be/tech-media/technologie/operer-un-patient-a-distance-c-est-desormais-possible-grace-a-la-5g/9958270.html>). Si le bras « se trompe » de quelques millimètres, le problème devient plus « sérieux ».

Cependant, dans le cas où les valeurs ne sont pas sous la forme de littéraux, mais mémorisées dans des variables, il faut passer par un opérateur particulier, l’opérateur de **conversion de type** ou ***cast***. Cet opérateur est un opérateur unaire puisqu’il porte sur un seul opérande.

La syntaxe est la suivante :

**(**<var\_type>**)**<var\_name>

Il faut y remplacer

* <var\_type> par un type, et
* <var\_name> par un nom de variable.

Si le total des points des AA proposés ci-dessus sont mémorisés dans la variable totalCotes et que le nombre d’AA de l’UE est dans nbCotes, l’instruction d’affichage de la moyenne de l’UE devient :

printf("Moyenne des cotes de l'UE : %f\n", (double)totalCotes / nbCotes);

Pour en revenir à la moyenne des jours ensoleillés, il faut procéder de la même façon.

Modifiez l’instruction que vous avez ajoutée dans le fichier source vacances.c de façon à afficher la bonne valeur pour un pourcentage.

printf("Pourcentage de jours ensoleilles sur les vacances : %f\n", …);

Ajoutez au fichier source torches.c l’instructions permettant d’afficher le pourcentage de torches qu’il vous reste dans votre inventaire par rapport au nombre de torches maximum.

Si vous avez repris les valeurs initiales (42 torches dans l’inventaire et un maximum de 64 torches), le résultat de ce calcul doit donner 65,625 %. N’oubliez pas que l’objectif de cet exercice est d’utiliser l’opérateur de conversion de type !

Cet opérateur a une priorité plus importante que tous les précédents.

## Opérateur « deux en un »

Vous savez que des variables peuvent être utilisées de part et d’autre de l’opérateur d’affectation =.

Il est fréquent de devoir modifier la valeur d’une variable sans pour autant vouloir la mémoriser sous un autre nom. Dans ce cas, on peut écrire

nbTorchesSac = nbTorchesSac + 22;

ou, pour gagner quelques caractères

nbTorchesSac += 22;

Ces deux instructions ont le même effet sur la variable nbTorches, elle l’augmente de 22.

Bien sûr cette façon de faire est applicable aux autres opérateurs. Vous aurez donc 5 nouveaux opérateurs à considérer : +=, -=, \*=, /= et %=.

Ces opérateurs ont une priorité moindre que les autres et seront donc à prendre en compte en dernier lieu.

Prévoyez ce que va afficher le programme suivant et vérifiez ensuite vos réponses…

int nbExercices = 7;

printf("nbExercices = %d\n", nbExercices);

nbExercices += 3;

printf("nbExercices = %d\n", nbExercices);

nbExercices \*= 4;

printf("nbExercices = %d\n", nbExercices);

nbExercices /= 5;

printf("nbExercices = %d\n", nbExercices);

nbExercices %= 3;

printf("nbExercices = %d\n", nbExercices);

## Incrémenter et décrémenter

Lorsque vous devez ajouter ou enlever 1 à une variable, il y a encore une autre façon de le noter. En effet, n = n + 1 peut s'écrire n += 1 mais les deux instructions suivantes ont le même effet :

* la **post-incrémentation** : n++
* la **pré-incrémentation** : ++n

Par exemple, après les instructions suivantes, n vaut 8 :

int n = 7;

n++;

Il en est de même après celles-ci :

int n = 7;

++n;

Si vous devez enlever 1 à une variable, vous pouvez le faire au moyen de

* la **post-décrémentation** : n--
* la **pré-décrémentation** : --n

Comme vous l’avez probablement deviné, ces opérateurs sont à nouveau des opérateurs unaires.

Ainsi après les instructions consécutives suivantes, n vaut 5 :

int n = 7;

n--;

--n ;

Il est cependant important de bien comprendre l’effet de ces instructions.

Dans l’exemple de code ci-dessous, quel sont, à votre avis, les valeurs des variables x et n après ces instructions ?

int n = 7;

int x = n++;

Créez un fichier source incrementation.c, et ajoutez-y les instructions permettant de vérifier que votre réponse correspond à l’exécution de ces instructions.

Assurez-vous de bien comprendre en vous aidant éventuellement d’un dessin pour visualiser ce qui se passe en mémoire. Il suffit de dessiner une case mémoire pour chacune des variables et de faire évoluer leurs valeurs au fur et à mesure des instructions.

Ajoutez maintenant les deux instructions suivantes :

n = 7;

x = ++n;

Quelle différence observez-vous ? Aidez-vous à nouveau d’un dessin si nécessaire !

Les instructions n++ et ++n ne sont pas seulement des instructions mais aussi des expressions. Ce sont des expressions un peu particulières parce qu’elles ont un **effet de bord**.

Une expression « normale » se contente de donner une valeur. Par exemple, 3 + 4 vaut 7 et nbJours \* 2 correspond au double de la valeur de nbJours.

Une expression à effet de bord donne une valeur et accomplit également quelque chose. Par exemple, dans

int x = 42;

int y = x++;

l’expression x++ donne une valeur qui est affectée à y et accomplit quelque chose de plus.

En toute généralité, un effet de bord (*side effect*) est le fait de provoquer quelque chose (un effet, un changement…) en plus de l'effet "principal"… C'est un effet secondaire.

Il en va de même pour la décrémentation, bien sûr !

L’effet des expressions x++ et ++x placés au sein d’instructions, comme par exemple y = ++x; ou printf("%d", x++); ne saute pas aux yeux.

Pour un code plus lisible, évitez d’utiliser des expressions x++ et ++x et contentez-vous de les utiliser comme instruction.

## Opérateur conditionnel

Comme vous l’avez déjà vu, il existe des opérateurs unaires, comme l’opérateur de transtypage, les opérateurs d’incrémentation et de décrémentation… et des opérateurs binaires comme l’opérateur d’affectation, les opérateurs arithmétiques et les opérateurs de comparaison.

Il existe également un opérateur ternaire car il fait intervenir 3 opérandes : l’**opérateur conditionnel**. Les expressions utilisant cet opérateur sont appelées **expressions conditionnelles**.

La syntaxe de cet opérateur est la suivante

<boolean\_expression> ? <expression> : <expression>

Il faut y remplacer

* <boolean\_expression> par une expression dont la valeur est booléenne, et
* <expression> par une expression, c’est-à-dire une valeur.

Si la valeur de l’expression booléenne vaut true, la valeur de l’expression conditionnelle est celle de la première expression, si elle vaut false la valeur de l’expression conditionnelle est la deuxième expression.

Par exemple, dans l’instruction

printf("Il a fait ensoleille durant toutes les vacances : %s\n",   
 nbJoursVacances == nbJoursVacancesEnsoleilles ? "vrai" : "faux");

si la valeur de l’expression booléenne est true, c’est-à-dire que la valeur de nbJoursVacancesEnsoleilles est égale à celle de nbJoursVacances, cela veut dire que l’affirmation «  Il a fait ensoleille durant toutes les vacances » est vraie, donc c’est la valeur « vrai » qui est affichée ; si la valeur de l’expression booléenne est false, c’est-à-dire que la valeur de nbJoursVacancesEnsoleilles est inférieure à celle de nbJoursVacances, c’est la valeur « faux » qui est affichée.

Complétez le fichier source torches.c avec les instructions suivantes. Remplacez les … par « vrai » ou « faux » afin d’afficher le résultat correspondant à l’affirmation.

int nbTorchesMax = 64;

int nbTorchesSac = 42;

printf("Le nombre de torches maximum est atteint %s\n",   
 nbTorchesSac == nbTorchesMax ? … : …);

printf("Le nombre de torches maximum est atteint %s\n",   
 nbTorchesSac != nbTorchesMax ? … : …);

printf("Le nombre de torches dans l’inventaire depasse la moitie de la capacite maximale en torches : %s\n",   
 nbTorchesSac <= nbTorchesMax / 2 ? … : …);

Voici le tableau des priorités des opérateurs, mis-à-jour avec ces derniers opérateurs :

|  |  |
| --- | --- |
| -(unaire) ++ -- (*type*) | ← |
| \* / % | → |
| + -(binaire) | → |
| < <= > >= | → |
| == != | → |
| ?: | ← |
| = += -= \*= /= %= | ← |