**Programmation**

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

**Bloc 1**

Exercice 0 – Variables, affichage et lecture

Objectifs

* S’approprier l’environnement de travail
* Décrire la notion de programme et sa structure en C
* Exécuter un programme simple
* Distinguer les éléments de base du langage : variable, type, littéral
* Employer les fonctions natives d’affichage et de lecture

[A. La ligne de commande de Visual Studio 3](#_Toc207876980)

[Configuration nécessaire 3](#_Toc207876981)

[Ouvrir une console 4](#_Toc207876982)

[Créer un fichier source 5](#_Toc207876983)

[Compiler un programme 7](#_Toc207876984)

[Exécuter un programme 9](#_Toc207876985)

[B. La structure d’un programme 10](#_Toc207876986)

[C. Avez-vous compris le début ? 11](#_Toc207876987)

[D. La notion de variable 12](#_Toc207876988)

[Affectation d’une valeur à une variable 12](#_Toc207876989)

[Déclaration d’une variable 13](#_Toc207876990)

[E. La notion de type primitif 15](#_Toc207876991)

[Mise en pratique 16](#_Toc207876992)

[F. La notion de littéral 17](#_Toc207876993)

[Les littéraux booléens 17](#_Toc207876994)

[Les littéraux entiers 17](#_Toc207876995)

[Les littéraux réels 17](#_Toc207876996)

[Les littéraux caractères 17](#_Toc207876997)

[Les littéraux chaîne de caractères 19](#_Toc207876998)

[G. L’affichage et la lecture 19](#_Toc207876999)

[Afficher à l’écran 19](#_Toc207877000)

[Mise en pratique 20](#_Toc207877001)

[Lire au clavier 20](#_Toc207877002)

[Mise en pratique 21](#_Toc207877003)

Dans ce document, plusieurs conventions sont utilisées. Elles sont décrites dans le tableau suivant.

|  |  |
| --- | --- |
| Mot en **gras** | Il s’agit d’un terme de vocabulaire (que vous devez donc maîtriser). |
| Cadre | Il s’agit de conventions de nommage, de règles de lisibilité, d’informations complémentaires… |
| Code | Il s’agit de bout de code à lire et à comprendre (et parfois à utiliser). |

# La ligne de commande de Visual Studio

Ce point présente les outils permettant d'écrire, de compiler et d'exécuter un programme C en utilisant la ligne de commande de Visual Studio.

Les objectifs pédagogiques liés à l’utilisation de la ligne de commande de Visual Studio sont les suivants :

* Définir et utiliser une console et une ligne de commande.

Interagir avec cette console au moyen des commandes de bases du shell.

* Distinguer les différentes étapes de la compilation d’un programme.
* Ne pas faire de vous des « presse-bouton » qui ne savent pas ce qui se cache derrière l’usuelle flèche  de la plupart des EDI – Environnement de Développement Intégré – comme Visual Studio ou Visual Studio Code.

## Configuration nécessaire

Système d’exploitation : Windows 10/11

Logiciels :

* un éditeur de texte. Par commodité, utilisez un éditeur de texte capable de coloration syntaxique pour le langage C, comme Notepad++ (<https://notepad-plus-plus.org>) ou Sublime Text (<https://www.sublimetext.com>).
* un compilateur qui respecte la norme C11. L’outil recommandé est Visual Studio avec les charges de travail de C++ (il y en a deux) : <https://docs.microsoft.com/fr-fr/visualstudio/install/install-visual-studio?view=vs-2022>

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Ces outils sont déjà installés sur les machines de l’école !

## Ouvrir une console

Pour accéder à l’invite de commande de la **console** proposée par Visual Studio, tapez « devel » dans la zone de recherche de Windows et choisissez le « Developer Command Prompt for VS 2022 » comme présenté sur la Figure 1.

Pour ceux qui ont l’interface en français : « Invite de commandes développeur pour VS 2022 ».

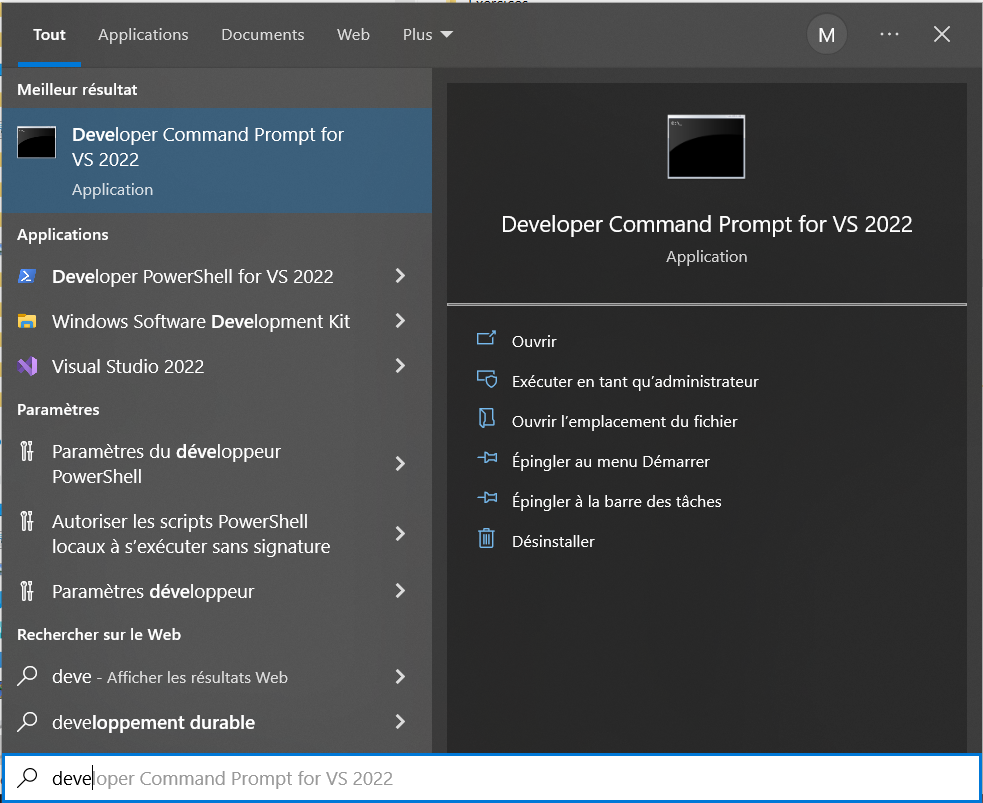


Figure 1 - invite de commande développeur

Une **console** ou **interface en ligne de commande** est une interface homme-machine dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode **texte** :

l'utilisateur tape une (ligne de) commande, c'est-à-dire du texte au clavier, en respectant des règles syntaxiques très précises ;

l'ordinateur affiche du texte correspondant au résultat de l'exécution de la commande.

Lorsque l'interface est prête à recevoir une commande, elle invite l'utilisateur à entrer sa commande au moyen de quelques caractères (le nom de compte de l'utilisateur, et/ou le chemin par défaut, et/ou date…) se terminant par un caractère particulier (**]**, **#**, $ ou **>**). Ces caractères forment ce qu'on appelle l'**invite de commande**.

Dans notre cas, l’invite de commande ressemble à ceci :

C:\Program Files (x86)\Microsoft Visual Studio\2022\Professional>

Vérifiez que la console est correctement configurée en tapant la commande cl comme montré ci-dessous.

C:\Program Files (x86)\Microsoft Visual Studio\2022\Professional>cl

Vous devriez obtenir quelque chose ressemblant à ce qui est présenté à la Figure 2. La version n’est probablement pas la même que celle sur l’image…

Une image contenant texte

Description générée automatiquement

Figure 2 - cl

Dans le cas contraire, il est probable que votre installation de Visual Studio ne s’est pas déroulée correctement.

## Créer un fichier source

Un **fichier** est un ensemble de données réunies sous un même nom. Un **fichier source** contient le **code source** d’un programme. En C, un fichier source a pour **extension** la lettre « c ». L’extension d’un fichier est un suffixe permettant d’identifier le format du fichier (.txt pour texte, .bmp pour Bitmap…).

Lancez votre éditeur de texte et créez un nouveau fichier.   
Pour cela, appuyez sur les touches CTRL + S pour sauver le fichier. Donnez-lui le nom bienvenue.c et sauvegardez-le sur le lecteur et dans le répertoire de votre choix.

Nom de fichier source

Le nom d’un fichier source est composé uniquement de caractères alphanumériques (les noms de fichiers ne sont pas sensibles à la casse), du tiret, du tiret-bas (*underscore*) et du point.

Dans cet exemple, le lecteur est u (lettre choisie au hasard, vous devriez avoir un lecteur c ou un lecteur d), le répertoire est Sources et le fichier est appelé bienvenue.c.

Lecteur et répertoire

De façon générale, un **répertoire** (*directory*) ou **dossier** (*folder*) est un ensemble de répertoires et/ou de fichiers.

L’ensemble des fichiers et répertoires forment une hiérarchie cohérente, appelée **arborescence**, dont le point d'entrée est le **répertoire racine**, qui est unique.

Il est également possible d’avoir plusieurs **lecteurs** sur un ordinateur. Historiquement, la lettre « a » est réservée au lecteur de disquettes permettant de charger le système d’exploitation et la lettre « b » à celui permettant de charger le logiciel à exécuter. De ce fait, il est d’usage de nommer le lecteur primaire avec la lettre « c ».

Il peut y avoir autant de lecteur que de disques (HDD, SSD…) voir même de partitions ou même de périphériques de lecture.

Recopiez le code source présenté ci-dessous dans le fichier bienvenue.c.

// Mon premier programme C

#include <stdio.h>

void main(void) {

printf("Bienvenue a l'Henallux !\n");

printf("Bienvenue au cours de programmation !");

}

Pensez à sauvegarder régulièrement votre travail !

## Compiler un programme

Si vous l’avez fermée, ouvrez la console de VS 2022 à nouveau. Déplacez-vous dans le répertoire contenant votre fichier source précédemment créé.

Naviguer dans l’arborescence des répertoires

Dans le cadre de console, vous n’avez pas les mêmes outils graphiques que lorsque vous êtes dans l’Explorateur de Windows .

Dans la console, vous allez devoir utiliser des **commandes**. Voici les quelques commandes dont vous risquez d’avoir besoin :

* changer de lecteur : tapez la lettre associée au lecteur suivie de « : », par exemple d:.
* changer de répertoire : utilisez la commande cd (*change directory*) suivie du chemin menant au répertoire cible, par exemple cd Sources pour aller dans le répertoire Sources, cd.. pour aller dans le répertoire « parent », cd \ pour aller dans le répertoire racine.
* afficher la liste des fichiers et sous répertoires : utilisez la commande dir (*directory*).
* Nettoyer la console : utilisez la commande cls (*clear screen*)

Si vous suivez l’exemple, vous devriez arriver à ce qui est présenté à la Figure 3.

Une image contenant texte

Description générée automatiquement

Figure 3 - se déplacer en ligne de commande

Une fois placé dans le répertoire, exécutez la commande suivante.

cl <source\_file>

où <source\_file> est le nom de votre fichier source (y compris son extension).

La Figure 4 montre ce qui est produit lors de l’exécution de la commande cl.

Une image contenant texte

Description générée automatiquement

Figure 4 - utilisation de cl

Si la compilation s’est correctement passée, un fichier .obj et un fichier .exe ont été créés.

Vérifiez que les deux fichiers générés au cours de la compilation… sont bien dans le répertoire au moyen de la commande dir.

Une image contenant texte

Description générée automatiquement

Figure - utilisation de dir

Il se peut que la compilation échoue si votre code source est incorrect. Le compilateur affiche alors des informations utiles sur la nature du problème. À vous de corriger les erreurs et de recompiler le code source.

Attention, tous les caractères (lettres, parenthèses, accolades…) sont nécessaires !

Indentation

L’indentation désigne l’ensemble des espaces, des tabulations et des passages à la ligne ajoutés pour faciliter la lisibilité du code source.

Elle n’est pas prise en compte par le compilateur… mais bien par le lecteur du code source en question !

Prêtez-y une attention toute particulière en respectant les normes précisées tout au long du cours.

## Exécuter un programme

Tapez simplement le nom du fichier .exe dans la console. La Figure 6 présente le résultat attendu.

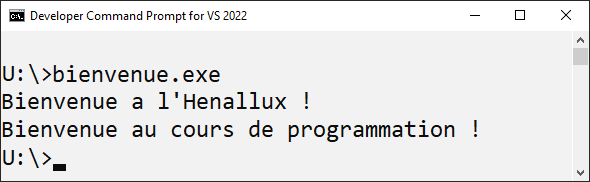


Figure 6 - affichage de bienvenue.exe

Vous pouvez aussi taper « b » (première lettre du nom de fichier bienvenu.exe) et appuyer sur la touche TAB, ainsi le nom du fichier est autocomplété. Lorsqu’il y a plusieurs fichiers qui commence par les mêmes lettres il prend le premier, donc à vous de mettre assez de lettres !

Une version complète de ce tutoriel est disponible à <https://docs.microsoft.com/fr-fr/cpp/build/walkthrough-compile-a-c-program-on-the-command-line>.

Toutes les options du compilateur et les informations sur celles-ci sont renseignées ici : <https://docs.microsoft.com/fr-fr/cpp/build/reference/compiler-options>.

# La structure d’un programme

Après avoir compilé et exécuté ce premier programme, il est intéressant de comprendre sa structure.

1 // Mon premier programme C

2 #include <stdio.h>

3

4 void main(void) {

5 printf("Bienvenue a l'Henallux !\n");

6 printf("Bienvenue au cours de programmation !");

7 }

La ligne (1) est une ligne de commentaires. Les commentaires ne sont pas pris en compte lors de la compilation, car ils n’ont aucun effet sur l’exécution du programme. Une autre façon de procéder pour mettre des commentaires est de les encadrer par /\* et \*/. Dans ce cas, ils peuvent être écrits sur plusieurs lignes comme ci-dessous.

/\* Mon

premier

programme C

\*/

La ligne (2) est une directive qui est traitée par le préprocesseur (un élément du compilateur). En effet, lors de la pré-compilation, le préprocesseur inclut les fichiers d’entête ou bibliothèques, d’extension .h (*header*), permettant l’utilisation de fonctions déjà définies.

Une **fonction** est un petit bout de code qui porte un nom et qui exécute une tâche précise. Ce concept est vu dans le Module 4.

Dans cet exemple, stdio.h contient la description de la fonction printf. Il s’agit en effet de la bibliothèque contenant les fonctions standards d’entrée (*input*) et sortie (*output*), d’où « std » pour standard, « i » pour *input* et « o » pour *output*.

La ligne (3) sert juste à aérer le code, elle n’est pas prise en compte par le compilateur.

Afin de rendre le code plus lisible, vous devez prendre l’habitude d’espacer le code, mais sans exagérer…

La ligne (4) est la ligne d’entête de la **fonction principale**. C’est le seul bloc d’instructions obligatoire. Il indique la partie du code qu’il faut exécuter lors du lancement du programme. On parle aussi de **point d’entrée du programme**.

Un **bloc d’instructions** est une suite d’instructions. Il est délimité par des accolades {…}.

Une **instruction** est un morceau de code qui correspond à une **action** : afficher un message, mettre une valeur dans une variable… Plus de détails sur la notion d’instruction sont présentés dans le Module 1.

La ligne (7) est celle qui permet de fermer le bloc d’instructions de la fonction principale.

Les lignes (5) et (6) sont des appels à une fonction déjà définie dans la bibliothèque stdio.h. Cette fonction permet d’afficher un message.

# Avez-vous compris le début ?

Commencez par créer un nouveau fichier appelé torches.c dans le même répertoire que celui dans lequel vous avez créé/sauvé le précédent.

Imaginez-vous en train de jouer à un jeu d’aventure sur votre tout nouveau PC. Dans votre sac à dos, souvent appelé inventaire dans les jeux, vous pouvez stocker divers objets.

Écrivez le programme qui permet d’afficher un message précisant le nombre de torches que vous avez dans votre inventaire. Le message ressemble à ceci :

J'ai 32 torches dans mon inventaire !

Lorsque vous l’avez écrit, compilez le programme. Pour ce faire, il faut à nouveau faire un appel à la commande cl en précisant que c’est le fichier torches.c qui doit être compilé.

Si le résultat de la compilation est une liste d’erreurs ou d’avertissements (*warnings*), corrigez-les. Dès que le code source est compilé correctement, exécuter le .exe qui en résulte. Vous devriez voir apparaître votre message à l’écran.

Ce programme doit correspondre à ceci :

#include <stdio.h>

void main(void) {

printf("J'ai 32 torches dans mon inventaire !");

}

# La notion de variable

En informatique, une variable associe un **nom** à une zone mémoire, repérée par son **adresse**, à laquelle on peut mémoriser une **valeur**. La Figure 7 en est une représentation simplifiée.

Figure - Variable

nom

valeur

adresse

Une variable correspond donc à une zone mémoire dans laquelle on stocke une valeur, une information, une donnée…

Quel que soit le langage, elle est caractérisée par

son **nom**, il est spécifié lors de sa déclaration,

son **type**, permet à l’ordinateur de réserver la zone mémoire adéquate (abordé plus loin),

sa **valeur** dont la représentation mémoire dépend du type de la variable,

son **adresse**, l’endroit où cette valeur est stockée en mémoire, définie par le système,

sa **portée**, la portion de code dans laquelle la variable est accessible.

## Affectation d’une valeur à une variable

L’instruction ci-dessous est une affectation ou assignation.

nbTorchesSac = 20;

L’**affectation** est une instruction qui a pour effet d’assigner une valeur à la variable, c’est-à-dire de placer la valeur en question dans la zone mémoire.

Dans cet exemple, elle a pour effet d’affecter la valeur 20 à la zone mémoire réservée pour la variable nbTorchesSac. Concrètement, par abus de langage, on dit souvent « mettre la valeur dans la variable ».

L’instruction d’affectation doit respecter la syntaxe suivante :

<var\_name> **=** <value>;

Il faut y remplacer

<var\_name> par le nom de la variable dont on veut modifier la valeur,

<value> par une valeur (au sens large).

Modifiez le programme torches.c de façon à intégrer cette instruction d’affectation et à afficher la valeur de la variable dans le message. Il faut également modifier l’appel de la fonction printf pour qu’elle prenne en compte cette variable. Elle devient printf("J’ai %d torches dans mon inventaire !", nbTorchesSac);.

Plus d’explications sur le printf sont données plus loin.

Le résultat de cette modification vous amène au programme ci-dessous.

1 #include <stdio.h>

2

3 void main(void) {

4 nbTorchesSac = 20;

5 printf("J'ai %d torches dans mon inventaire !", nbTorchesSac);

6 }

Compilez ce programme et observez le résultat de la compilation dans la console.

Vous devriez avoir obtenu les messages présentés à la Figure 8.



Figure - Une erreur torches.c

Ces messages d’erreur précisent que, à la ligne (4) du programme ainsi qu’à la ligne (5), l’identificateur nbTorchesSac n’est pas déclaré.

Un **identificateur** ou **identifiant** est un mot qui désigne un élément du programme : variable, constante, procédure, fonction, type, etc.

## Déclaration d’une variable

En C, une variable doit être déclarée avant de se voir affecter une valeur.

La **déclaration** est l’instruction qui permet de préciser le **type** de la variable, c’est-à-dire le type des valeurs qui pourront lui être affectées par la suite, ainsi que son **nom**. C’est à l’exécution de cette instruction qu’un espace mémoire, correspondant au type, est réservé/alloué et associé au nom de la variable.

Pour déclarer une variable il faut donc respecter la syntaxe suivante :

<var\_type> <var\_name>;

Il faut y remplacer

<var\_type> par le type que la variable doit avoir,

<var\_name> par le nom de la variable.

Corrigez le code du programme torches.c de façon à déclarer la variable nbTorchesSac. En toute logique, le nombre de torches ne peut être qu’un entier. On utilise donc le type int (*integer*).

Vous devez donc ajouter l’instruction suivante avant l’affectation.

int nbTorchesSac;

Compilez à nouveau ce programme. Les fichiers .obj et .exe correspondant ont normalement été générés. Exécutez le programme !

Modifiez la valeur de la variable dans le code source (**sans recompiler**) et exécutez le programme. Que pouvez-vous observer ?

Compilez une dernière fois ce programme et exécutez-le. Le message affiché a-t-il été mis à jour ?

Assurez-vous de bien comprendre la démarche dans son ensemble !

Nom de variable

Le nom d’une variable est un nom que vous choisissez vous-même assez librement.

En C, les seules règles à respecter au niveau syntaxique sont les suivantes :

* il est composé d'au moins une lettre et peut également comporter des chiffres et/ou le caractère \_
* sa longueur ne peut pas dépasser 32 caractères
* le premier caractère doit être une lettre ou \_

Cependant, efforcez-vous de bien les choisir : de préférence assez courts, mais aussi explicites que possible, de manière à exprimer clairement ce que la variable est censée contenir.

Par exemple, des noms de variables tels qu’année et mois conviennent mieux qu’a et m, car un programmeur doit faire en sorte que les instructions soient faciles à lire.

De plus, il est recommandé d'écrire les noms de variables en commençant par une minuscule.

Si le nom d'une variable doit comporter plusieurs mots, il y a de nombreuses possibilités pour écrire le nom de la variable. Parmi celles-ci, on retrouve souvent :

* le camelCase, c'est-à-dire en séparant les mots par un passage en majuscule.   
  Exemple : margeBrute.
* le snake\_case, c'est à dire en séparant les mots par le caractère "\_".   
  Exemple : marge\_brute.

Il convient aussi d'éviter autant que possible l'énumération de variables (toto1, toto2, toto3...), cela rend le programme parfaitement incompréhensible et sujet à des erreurs.

Il est possible de préfixer le nom de la variable par son type. Par exemple int\_margeBrut, str\_message\_de\_bienvenue, mais cela alourdit très fortement le programme.

D'autres notations existent ou peuvent être imposées en fonction d'un projet, des habitudes d'une entreprise, etc.

Dans le cadre de la plupart des cours de programmation du bloc 1, c’est la notation en **camelCase** qui est préférée !

# La notion de type primitif

Une valeur peut prendre différentes formes : un entier, un réel, un caractère, une chaine de caractères, une adresse, voire une expression (qui doit être évaluée avant de pouvoir être utilisée) …

Parmi ces formes, certaines sont dites primitives ou basiques ou primaires, et d’autres sont plus complexes et seront abordées dans la suite du cours.

Le **type** d’une variable correspond à l’ensemble des valeurs admises, c’est-à-dire qu’elle peut contenir. Il définit la façon dont cette valeur est représentée en mémoire.

Les types primitifs sont donc les suivants :

pour les **entiers** : short int ou short, **int**, long int ou long

pour les **réels** : float, **double**, long double

pour les **caractères** : char

pour les **booléens** : bool 🡪 dans la bibliothèque stdbool.h

*La représentation en mémoire des valeurs correspondant à ces types n’est pas détaillée dans ce document, mais sa compréhension reste utile à l’utilisation des bons types.*

Vous avez déjà l’habitude de manipuler des valeurs entières, réelles et même les caractères, mais les booléens vous sont peut-être un peu moins familiers.

En effet, le type bool correspond aux valeurs true et false.

Concrètement, dans un formulaire, si vous désirez savoir si un étudiant est dans le bloc 1, vous pouvez proposer un choix sous la forme d’un petit carré à sélectionner comme à la Figure 9.



Figure - Exemple de booléens

Si le carré associé au bloc 1 n’est pas sélectionné, c’est que l’étudiant n’est pas dans le bloc 1, l’affirmation « J’ai des cours dans le bloc 1 » est fausse ; si le carré est sélectionné, l’affirmation « J’ai des cours dans le bloc 1 » est vraie.

Pour mémoriser cette information sous la forme d’une variable, le plus simple est donc d’utiliser une variable estDansBloc1. Dans le cas où l’étudiant a des cours dans ce bloc, sa valeur est true, et si non sa valeur est false. Il n’y a pas d’autres valeurs possibles.

## Mise en pratique

1. Voici quelques informations qui pourraient être nécessaires dans un programme. Choisissez pour chacune d’elles un nom de variable adéquat et le type le plus approprié pour les mémoriser.

|  |  |  |
| --- | --- | --- |
| Informations | Noms de variable | Type |
| Date au format AAAAMMJJ | dateYearsMounthDay | Tab de char |
| Surface en m2 du sol d’une chambre de kot | floorAreaStudentBedroom | Unsigned short |
| Nombre de jours sans jouer sur son PC | dayWithoutPlaying | Unsigned int |
| Première lettre de votre nom de famille | firstFamilyNameChar | char |
| Avoir un permis de conduire | haveDrivingLicense | Bool |
| Âge de l’étudiant(e) le(la) plus jeune | youngestStudentAge | Unsigned char |

N’hésitez pas à discuter de vos propositions avec d’autres étudiants… C’est une bonne façon de faire connaissance et d’échanger vos points de vue.

1. Écrivez chacune des instructions suivantes (en respectant l’ordre) dans un nouveau fichier nommé pekets.c :
   1. déclarez une variable permettant de mémoriser le nombre de peket(s) que vous désirez commander.
   2. affectez le nombre entier de votre choix à cette variable.
2. Écrivez chacune des instructions suivantes (en respectant l’ordre) dans un nouveau fichier nommé lettreGroupe.c :
   1. déclarez une variable permettant de mémoriser la lettre de votre groupe.
   2. affectez la lettre de votre choix à cette variable.
3. Écrivez chacune des instructions suivantes (en respectant l’ordre) dans un nouveau fichier nommé aPayer.c :
   1. déclarez une variable permettant de mémoriser un montant en euros, correspondant au montant à payer.
   2. affectez le nombre réel de votre choix à cette variable.

# La notion de littéral

À chacun de ces types primitifs correspondent diverses valeurs. L’écriture de ces valeurs doit respecter certaines règles précises. On appelle ces valeurs, des **littéraux**.

Une **valeur littérale** ou un **littéral** est une valeur donnée explicitement dans le code source d'un programme : 3, 4.5, true, "Bienvenue".

Le format à utiliser dépend de son type : 3 et 3.0 ne sont pas les mêmes littéraux.

## Les littéraux booléens

Les deux seuls littéraux booléens (de type bool) sont true et false.

## Les littéraux entiers

Les littéraux de type entier peuvent être exprimés de différentes façons. Par défaut, ils sont de type int et exprimés en base décimale : 430 -79 0

## Les littéraux réels

Les littéraux de type réel peuvent contenir le séparateur décimal anglophone, c’est-à-dire le point, et éventuellement un exposant.

Par défaut, ils sont de type double et sont toujours exprimés en base décimale :   
0. 1.57 3.14e+5 34.6E-3

⮱ 3.14\*105 ⮱ 34.6\*10-3

## Les littéraux caractères

Les littéraux de type caractère étant représentés sous la forme d’un nombre en mémoire, plusieurs possibilités s’offrent à vous pour les exprimer. Cependant, la façon la plus lisible est d’écrire simplement le caractère entre apostrophes : 'a' 'A' '3' '+'

Code ASCII

Le code ASCII est un nombre mémorisé sur 8 bits. Il s’agit donc d’une valeur entière qui, en base 10, doit être comprise entre 0 et 255. À chacune de ces valeurs correspond un caractère : 'A' est représenté par 6510 ou 4116. L’ensemble des codes ASCII est regroupé dans ce qui est communément appelé la table ASCII.

Pour des raisons évidentes de lisibilité du programme que vous rédigé, il est fortement déconseillé d’utiliser le code ASCII d’un caractère directement dans le programme.

Imaginez un code dans lequel vous avez besoin d’afficher le caractère 'A', pensez-vous que ce code serait compréhensible si vous écrivez l’instruction (1) ou l’instruction (2) ?

(1) printf("%c", 65);

(2) printf("%c", 'A');

Aucun développeur n’est supposé avoir appris la table ASCII par cœur… De plus les 128 derniers éléments changent selon la langue !

## Les littéraux chaîne de caractères

Bien que les variables permettant de contenir une chaîne de caractères sont de type non-primitif ou dérivé, vous aurez régulièrement besoin de tels littéraux pour afficher des messages à l’écran. Vous ne pouvez pas encore les mémoriser !

Un littéral de type chaîne de caractères est une suite de caractères entourée de guillemets : "Bonjour !" "\tMonsieur,\n" "1234" "e"

# L’affichage et la lecture

Les instructions d’entrée/sortie (**i**nput/**o**utput) permettent de dialoguer avec l’utilisateur.

Les instructions permettant d’afficher quelque chose à l’écran ou de l’envoyer à l’imprimante ou tout autre périphérique de sortie, s’appellent « instructions de sortie ».

Celles permettant de récupérer une information provenant d’un clavier, de la souris ou d’un autre périphérique d’entrée, s’appellent « instructions d’entrée ».

Il est d’usage d’appeler ces deux types d’instructions des « instructions d’entrée/sortie », d’où, comme vous le savez maintenant le nom de la bibliothèque permettant leur utilisation : stdio.h.

Ces instructions sont réalisées en C grâce à des fonctions telles que printf, scanf, getchar, putchar, gets, puts, fread, fwrite… Seules les deux premières sont détaillées dans ce document. Les autres seront abordées dans la suite du cours voire au Q2.

Dans notre cas, la fonction printf permet d’afficher à l’écran alors que la fonction scanf permet de lire ce qui est entré au clavier.

## Afficher à l’écran

Vous avez déjà utilisé la fonction printf de deux façons :

pour afficher un message (littéral) :

printf("Bienvenue a l'Henallux !");

// et

printf("J'ai 32 torches dans mon sac a dos !");

pour afficher un message intégrant une valeur de variable :

printf("J'ai %d torches dans mon sac a dos !", nbTorches);

Il existe en effet plusieurs façons d’utiliser cette fonction pour afficher des informations qu’elles soient littérales ou non, et même de mélanger les deux…

En toute généralité, la fonction printf suit la syntaxe suivante :

printf("<write\_format>", <var\_name>);

Il faut y remplacer

<write\_format> par une chaîne de caractères comportant éventuellement le(s) format(s) d’affichage correspondant au type de la (des) variable(s) à afficher,

<var\_name> par le(s) éventuel(s) nom(s) de variable(s) dont vous désirez afficher la valeur.

Selon le type de la valeur à afficher, le format à utiliser dans la chaîne de caractères est différent. Voici les principaux formats en question :

|  |  |
| --- | --- |
| Type | Format d’affichage |
| int | %d |
| double | %f |
| char | %c |
| chaîne de caractères | %s |

## Mise en pratique

1. Modifiez le programme pekets.c de façon à afficher le message « Je voudrais XX peket(s), s’il-vous-plait » en remplaçant XX par le nombre de peket(s) mémorisé dans la variable.
2. Modifiez le programme lettreGroupe.c de façon à afficher le message « Votre groupe est XX » en remplaçant XX par la lettre mémorisée.
3. Modifiez le programme aPayer.c de façon à afficher le message « Vous devez payer XX euros » en remplaçant XX par la valeur mémorisée.

## Lire au clavier

Pour pouvoir récupérer des informations entrées par l’utilisateur au clavier, vous devez utiliser la fonction scanf.

En toute généralité, la fonction scanf suit la syntaxe suivante :

scanf("<read\_format>", **&**<var\_name>);

Il faut y remplacer

<read\_format> par une chaîne de caractères comportant éventuellement le(s) format(s) de lecture correspondant au type de la (des) valeur(s) à récupérer,

<var\_name> par le(s) éventuel(s) nom(s) de variable(s) auxquelles vous désirez affecter cette(ces) valeur(s).

Selon le type de la valeur à lire, le format à utiliser dans la chaîne de caractère est différent. Voici les principaux formats en question :

|  |  |
| --- | --- |
| Type | Format de lecture |
| int | %d |
| double | %**l**f |
| char | %c |

Pour demander le nombre de torches à l’utilisateur, vous pouvez écrire

printf("Nombre de torches dans l'inventaire : ");

scanf("%d", **&**nbTorches);

## Mise en pratique

1. Modifiez le programme pekets.c de façon à demander le nombre de pekets à l’utilisateur et à le récupérer dans la variable déclarée.
2. Modifiez le programme encore.c de façon à demander la lettre à l’utilisateur et à le récupérer dans la variable déclarée.
3. Modifiez le programme aPayer.c de façon à demander le montant à l’utilisateur et à le récupérer dans la variable déclarée.