**Langage de 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

[Introduction 2](#_Toc81224019)

[A. Utiliser la ligne de commande de Visual Studio 2](#_Toc81224020)

[B. Structure d’un programme 9](#_Toc81224021)

[C. Avez-vous compris le début ? 10](#_Toc81224022)

[D. La notion de variable 11](#_Toc81224023)

[E. La notion de type primitif 14](#_Toc81224024)

[F. La notion de littéral 16](#_Toc81224025)

[G. Affichage et lecture 17](#_Toc81224026)

[H. Du problème au programme 19](#_Toc81224027)

# Introduction

Cette série d’exercices est à réaliser lors des 2 premières séances de théorie et en colearning, en préparation du premier cours de théorie.

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*).

# 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 (voir cours de « Systèmes d’exploitation »).

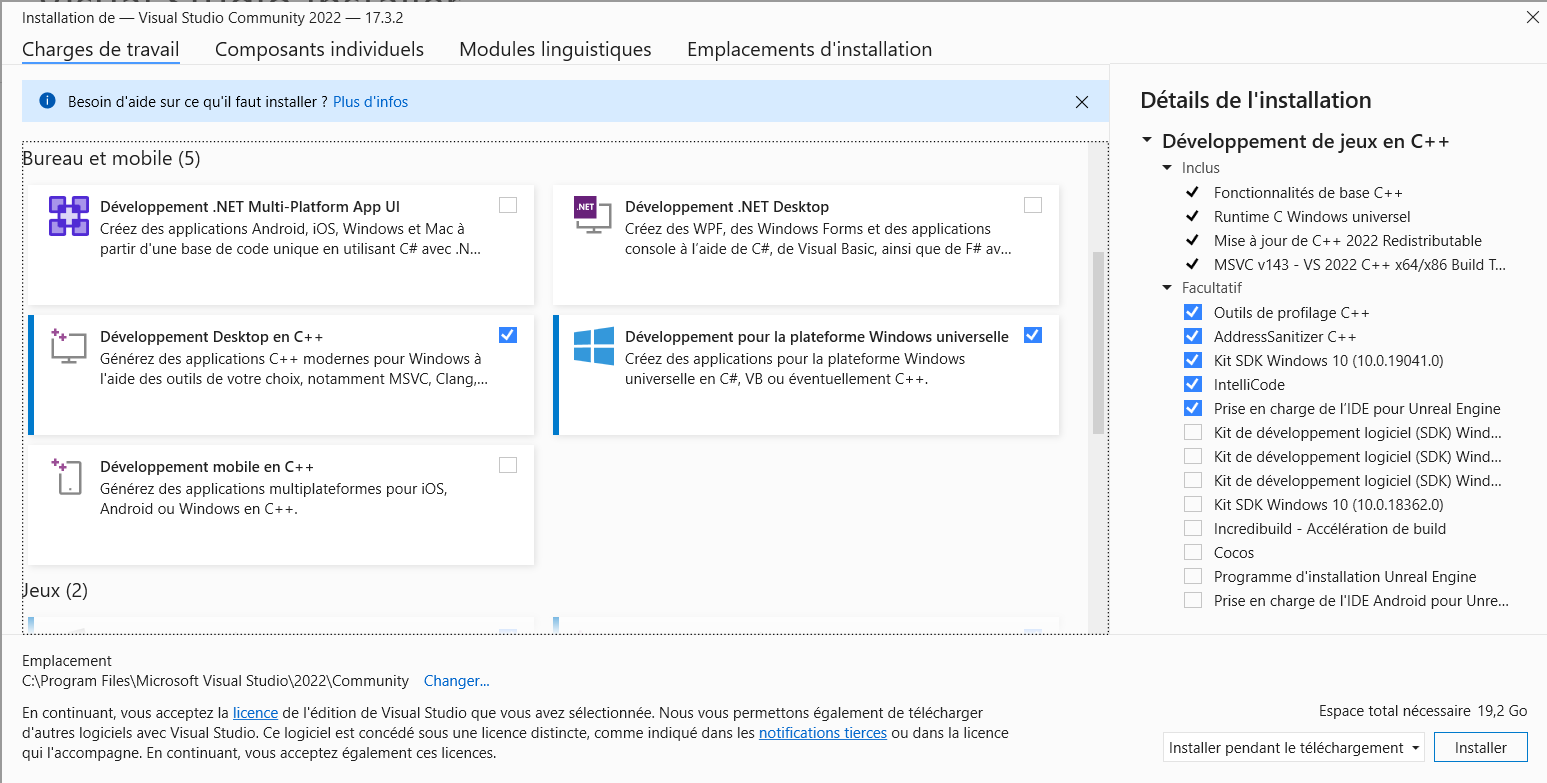
* Distinguer les différentes étapes de la compilation d’un programme.
* Ne pas faire de vous des « presse-bouton » qui ne savent pas ce qui se cache derrière l’usuelle flèche verte  de la plupart des EDI – Environnement de Développement Intégré – comme Visual Studio (ou IntelliJ que vous utiliserez pour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

## 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>).
* **soit** 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>



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 ».

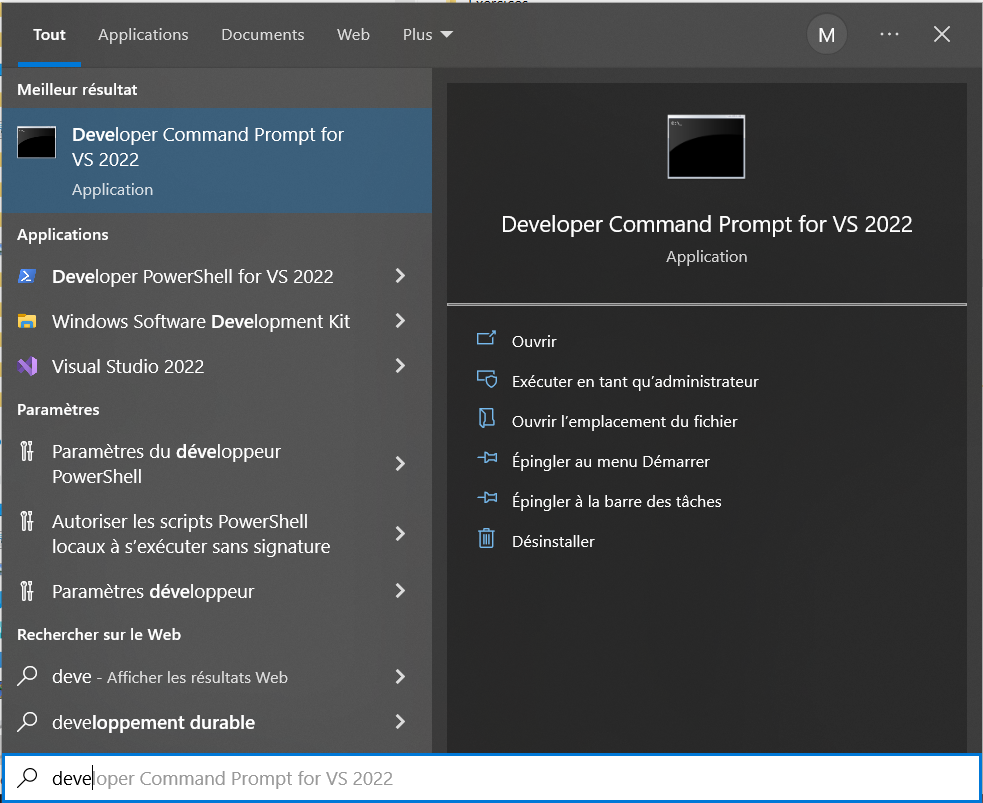


Figure 1 - invite de commande développeur

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 un éditeur de texte que vous avez choisi d’utiliser et créez un nouveau fichier appelé bienvenue.c dans lequel écrire votre code source.

Recopiez le code source présenté ci-dessous (ou proposé par votre professeur de labo qui est probablement plus inspiré…).

1 // Mon premier programme C

2 #include <stdio.h>

3

4 void main(void) {

5 printf("Bienvenue à tous !\n");

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

7 }

Ce programme affiche les messages « Bienvenue à tous ! » et « Bienvenue au cours de Langage de programmation ! ».

Sauvegardez votre fichier source sur le lecteur, dans le répertoire et sous le nom de votre choix.

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.

Dans cet exemple, le lecteur est u (lettre associée à votre compte sur le réseau de l’école), le répertoire est Sources et le fichier est appelé bienvenue.c.

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.

## 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 u:.
* 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

Plus d’informations sur ces deux extensions et sur la création des fichiers concernés sont données au cours de théorie et au cours d’« Architecture des ordinateurs ». Mais pour les plus impatients, quelques explications sont données en fin de document.

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.   
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 !

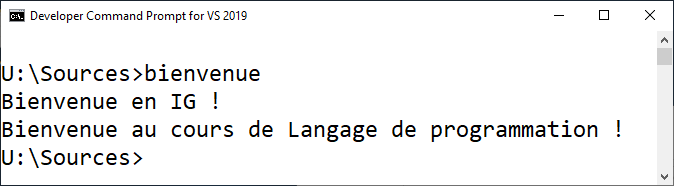


Figure 6 - affichage de bienvenue.exe

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>.

# 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 à tous !");

6 printf("Bienvenue au cours de Langage 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.

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.

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.

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

# Avez-vous compris le début ?

Afin de vérifier que vous avez bien compris l’utilisation de la ligne de commande ainsi que la façon de rédiger un programme simple, un exercice s’impose.

Commencez par créer un nouveau fichier appelé torches.c dans le même répertoire que celui dans lequel vous avez créé/sauvé le précédent. Plusieurs solutions s’offrent à vous, à vous de choisir celle qui vous convient le mieux.

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, compiler 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 :

2 #include <stdio.h>

3

4 void main(void) {

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

7 }

# La notion de variable

En informatique, une variable associe donc 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.

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 ci-dessus (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.

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.

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 !", nbTorches);

6 }

Convention interne

Dans le cadre de la plupart des cours de programmation que vous allez suivre en IG, c’est la notation en **camelCase** qui est préférée !

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 nbTorches n’est pas déclaré. On parle d’identificateur car les noms de variable font partie de cette catégorie d’éléments du langage.

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

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 nbTorches. En toute logique, le nombre de torches ne peut être qu’un entier. On utilise donc le type int (*integer*).

Le programme résultant de cette correction est le suivant.

1 #include <stdio.h>

2

3 void main(void) {

4 int nbTorches;

5 nbTorches = 20;

6 printf("J’ai %d torches dans mon inventaire !", nbTorches);

7 }

Compilez à nouveau ce programme. Les fichiers .obj et .exe correspondant ont normalement été générés. Vous pouvez donc exécuter 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 a-t-il été mis à jour ?

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

# La notion de type primitif

Une valeur peut prendre différentes formes : un entier, un réel, un caractère, une chaine de caractères, une adresse, voire même 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 est vue en profondeur dans le cours d’« Architecture des ordinateurs ».

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.

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 | date | Char[] |
| Surface en m2 du sol d’une chambre de kot | surfaceSol | double |
| Nombre de jours sans jouer sur son PC | nbrsJoursSansPc | Int |
| Première lettre de votre nom de famille | premiereLettreNom | Char |
| Avoir un permis de conduire | aLePermis | Bool |
| Âge de l’étudiant(e) le(la) plus jeune | ageEtudiantPlusJeune | float |

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.

Ci-après, quelques exercices vous sont proposés. Veillez à bien suivre les consignes car vous allez les réutiliser dans la suite de l’atelier…

Écrivez chacune des instructions suivantes (en respectant l’ordre) dans un nouveau fichier nommé pekets.c :

déclarez une variable permettant de mémoriser le nombre de peket(s) que vous désirez commander.

initialisez cette variable avec le nombre de votre choix.

Écrivez chacune des instructions suivantes (en respectant l’ordre) dans un nouveau fichier nommé encore.c :

déclarez une variable permettant de mémoriser un caractère, correspondant à la réponse (‘o’ pour oui, ‘n’ pour non) à une question.

affichez le message « Voulez-vous encore commander des pekets ? ».

affectez la valeur de votre choix à la variable : 'o' ou 'n'.

Écrivez chacune des instructions suivantes (en respectant l’ordre) dans un nouveau fichier nommé aPayer.c :

* déclarez une variable permettant de mémoriser un montant en euros, correspondant au montant à payer.
* affectez la valeur de votre choix à la variable.
* affichez le message « Martin doit payer … euros », en remplaçant les « … » par la valeur de la 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 exprimé 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' '+'

## 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"

# Affichage et 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 à tous !");

// et

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

pour afficher un message intégrant une valeur de variable :

printf("J'ai %d torches dans mon sac à 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 |

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.

Modifiez le programme encore.c de façon à afficher le message « Votre réponse est XX » en remplaçant XX par le caractère mémorisé.

Modifiez le programme aPayer.c de façon à afficher le message « XX doit payer YY euros » en remplaçant XX par l’initiale de votre prénom et YY par la valeur mémorisée.

## Lire au clavier

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

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

scanf\_s("<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\_s("%d", **&**nbTorches);

Dans le cas d’un caractère, un élément supplémentaire doit être noté : le nombre de caractères demandés. Dans un premier temps, comme vous ne pouvez pas encore obtenir de valeur sous la forme d’une chaîne de caractères, ce nombre est toujours à 1.

printf("Groupe (A, B, C, D ou E) :");

scanf\_s("%c", &groupe, 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.

Modifiez le programme encore.c de façon à demander la réponse à l’utilisateur et à le récupérer dans la variable déclarée.

Modifiez le programme aPayer.c. de façon à demander le montant à l’utilisateur et à le récupérer dans la variable déclarée.

# Du problème au programme

Afin de comprendre la démarche effectuée pour exécuter votre premier programme, quelques notions doivent être précisées.

## Du problème au code source

Lorsqu’on veut faire en sorte de déléguer une tâche à l’ordinateur, il faut d’abord réfléchir à ce qui la compose, aussi bien en termes de données qu’en termes d’actions. C’est seulement ensuite qu’on peut passer à la rédaction du programme.

Le passage de l’idée qu’on se fait du problème à sa représentation sous la forme d’un « programme » suit deux étapes principales : la **conception**, qui est la démarche vue dans les cours de « Principes de programmation » et le **codage**, qui est la démarche vue dans ce cours-ci.

La phase de **conception** est indépendante du langage de programmation utilisé. C’est une phase de réflexion qui aboutit à la rédaction d’un **pseudo code** (langage simplifié dont l’objectif est de formaliser la tâche à réaliser sans s’encombrer des éléments techniques et syntaxiques liés à l’ordinateur).

La phase de **codage** est directement liée au langage de programmation utilisé. Elle consiste à traduire le pseudo code dans un ***langage cible***, ici le C. Au terme de cette phase, le programmeur a rédigé un **code source**.

La Figure 10 présente les phases de conceptions et de codage, ainsi que les deux phases qui suivent.

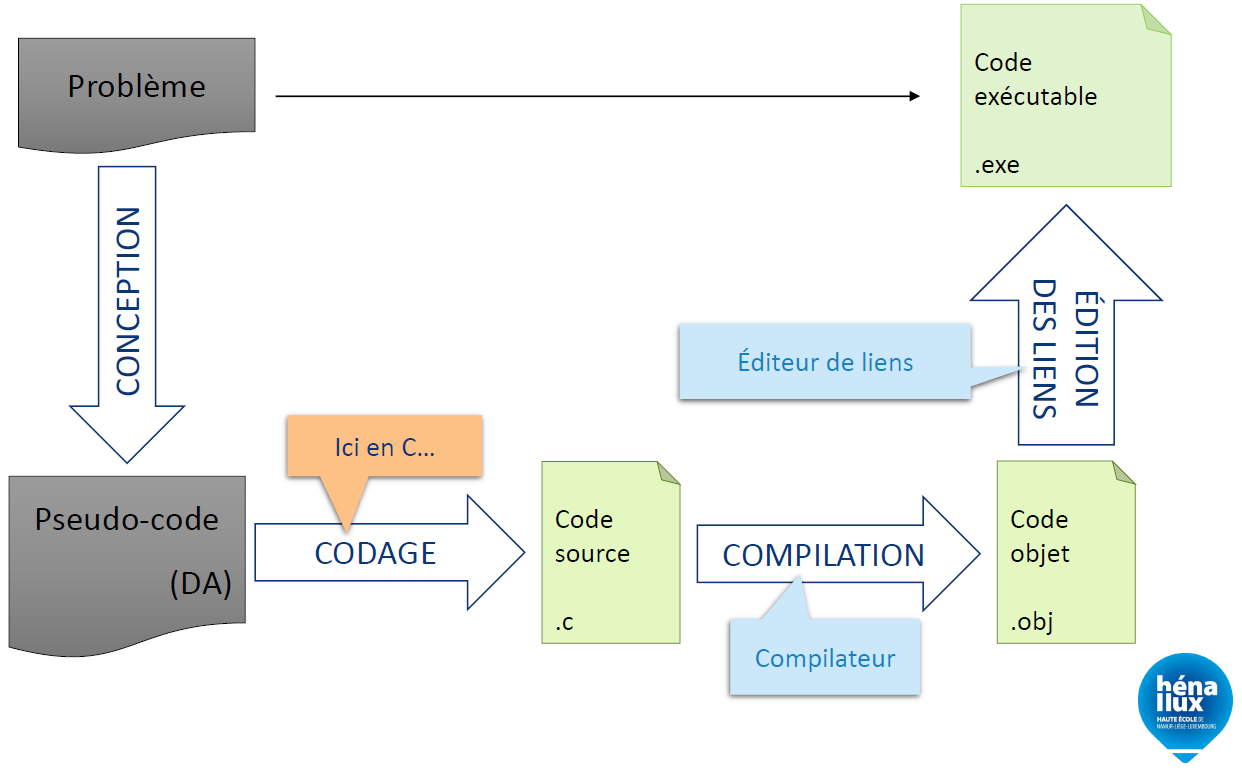


Figure - Du problème au programme exécutable

## Du code source au programme exécutable

Lorsque le programmeur a rédigé le code source et s’est assuré qu’il répond bien au problème initial, il est temps de demander l’avis de l’ordinateur. Pour cela, deux phases s’ajoutent aux précédentes : la **compilation** et l’**édition des liens**. Comme le montre la Figure 11, ces deux phases sont décomposées en plusieurs étapes qui sont vues de façon un peu plus détaillée en « Architecture des ordinateurs » (UE123).

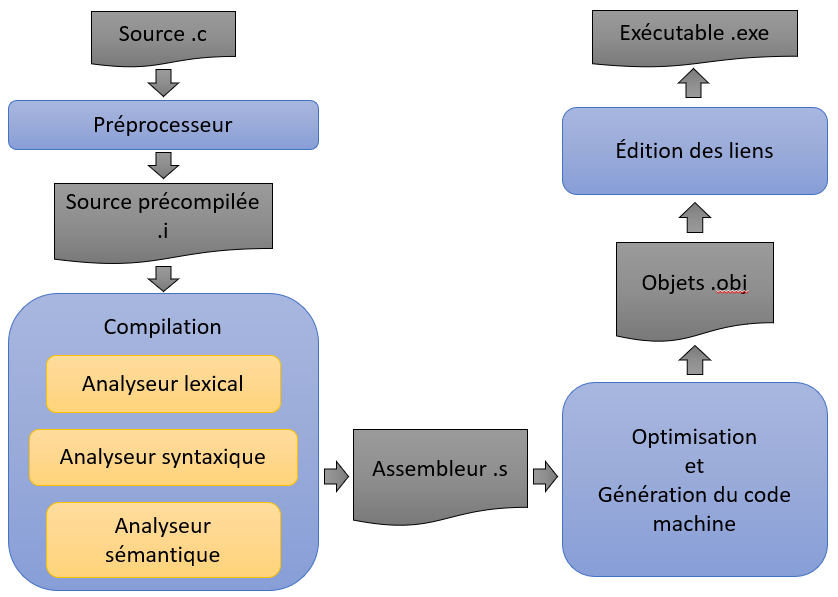


Figure - Étapes de la compilation et de l'édition des liens

La **compilation** permet de traduire le code source dans un langage plus proche de l’architecture de l’ordinateur. Lors de cette étape, plusieurs optimisations peuvent également être apportées par le compilateur.

L’**édition des liens** permet de lier les différents programmes objets résultant de la compilation ainsi que les fichiers d’entêtes et autres bibliothèques.

Le fichier **source** (.c), doit être compilé pour donner un fichier **objet (**.obj). Il faut ensuite que l’éditeur de lien fasse son travail d’intégration des fichiers objets pour générer le fichier **exécutable** (.exe).