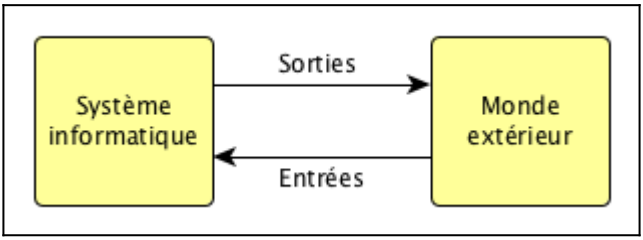
Introduction sur les entrées et sorties

Les entrées et sorties en informatique

D'une manière générale, les entrées et sorties d'un système informatique quelconque correspond aux échanges d'informations entre ce système et le monde extérieur (c'est à dire tout ce qui n'est pas ce système). Les termes “entrée” et “sortie” doivent se comprendre par rapport au système considéré, c'est à dire qu'une entrée correspondent aux informations qui entrent dans le système, et les sorties aux informations qui sortent de ce système.



En anglais, le terme “entrée et sortie” se traduit par *input-output*, qui est souvent abgree en *IO*.

La nature des informations échangées et les interlocuteurs du système vont dépendre du système considéré. Par exemple, si vous considerez le processeur central d'un ordinateur, les entrées-sorties vont être des signaux électriques échangées avec les autres composants électroniques d'un ordinateur (en particulier les mémoires : mémoire vive, disques durs, etc).

Si vous considerez un ordinateur dans son ensemble, les entrées-sorties vont être les informations échangées avec l'environnement. Cela peut

être les interactions avec les utilisateurs via les périphériques (clavier, souris, écran). Cela peut être également les communications avec d'autres système informatique, via un réseau informatique (cable reseau, WiFi, 4G, etc).

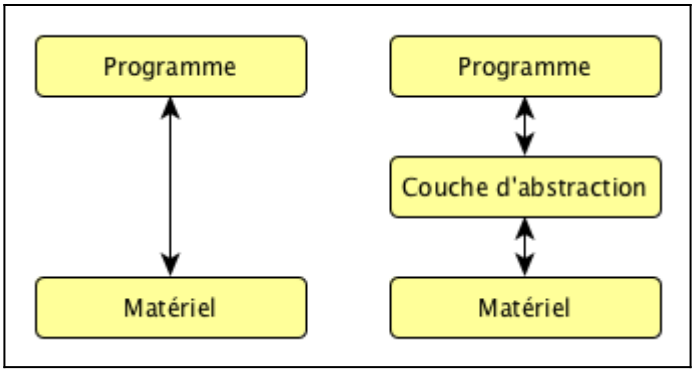
Couches d'abstraction

D'un point de vue fondamental, toutes les communications informatiques sont des informations électriques dans les composants électroniques qui composent un système informatique. Mais il serait très compliqué de concevoir les programmes en travaillant uniquement sur cette vision très bas niveau de l'informatique. Cette approche était utilisée dans les premiers temps de l'informatique, mais a montrée ses limites dans la conception de systèmes complexes.

Prenez par exemple le problème du matériel informatique. Chaque ordinateur utilise des composants spécifiques, avec une architecture différentes selon le fabricant. Ecrire un code spécifiquement pour un type de matériel implique donc de devoir écrire un programme différent pour chaque type d'ordinateur. Cela devient impossible de nos jours, ou il existe des millions d'architectures différentes.

La solution est de ne pas écrire un programme en utilisant un code spécifique d'un matériel particulier, mais de créer un intermédiaire, qui fera la transition avec le matériel. Le programme utilisera uniquement cet intermédiaire, ce qui implique que le code du programme ne changera pas en fonction du matériel. Bien sur, cet intermédiaire devra être écrit spécifiquement selon le matériel, mais il sera utilisé par tous les programmes.

Cet intermédiaire est appelée une couche d'abstraction. Il permet de masquer tout ce qui se passe en dessous et propose une interface commune (abstraction) pour tous les programmes qui utilisent cette couche.



Ce concept de couches peut être étendu autant de fois que nécessaire pour simplifier la programmation. Par exemple, pour concevoir un programme qui utilisent Internet, il est possible d'écrire une couche pour le matériel, une couche pour le réseau, une couche pour le transport des données, une couche pour l'identification sur le réseaux, etc. Un exemple de telle structure en couche pour la communication est le modèle OSI (de l'anglais *Open Systems Interconnection*), qui contient sept couches.

Il est beaucoup plus simple de concevoir un programme de cette façon. Chaque couche ne doit être pensée qu'en prenant en compte que deux autres couches :

- la couche en dessous, que vous utilisez pour écrire la couche sur laquelle vous travaillez ;
- la couche au dessus, qui utilisera la couche sur laquelle vous travaillez.

Plus une couche est proche du matériel, plus elle est de bas-niveau. Plus elle s'en éloigne, plus elle est de haut-niveau. La couche la plus proche du matériel est appelée “couche d'abstraction du matériel” (*HAL* en anglais, pour *Hardware Abstraction Layer*).

Bibliothèques logicielles et modules

A ce niveau, vous vous posez peut être une question : c'est bien beau

cette architecture en couche, mais s'il faut réécrire les couches a chaque fois que vous écrivez un programme, quel est l'intérêt ?

Et bien sur, la reponse est evidente : vous n'avez pas besoin de réécrire toutes les couches a chaque fois. De nombreuses couches (ou parties de couches) sont disponibles par défaut (ou facilement disponibles), il vous suffit de les utiliser.

C'est justement le rôle des bibliothèques logicielles de fournir des fonctionnalités réutilisables dans les programmes. Vous avez déjà par exemple utiliser la bibliothèque standard du C++, qui fournit des fonctionnalités comme `std::cout` pour afficher du texte à l'écran. Vous n'avez pas besoin de savoir comment fonctionne un écran d'ordinateur en interne, vous avez juste besoin de savoir comment utiliser `std::cout` pour afficher du texte.

Pour les fonctionnalités autres que celle fournie par le système d'exploitation et par la bibliothèque standard du C++, vous avez à disposition un nombre important de bibliothèques externes. Vous trouverez par exemple une liste de telles bibliothques sur cppreference.com : http://en.cppreference.com/w/cpp/links/libs.

Un module est un ensemble cohérent de fonctionnalités. Par exemple, vous pouvez avoir dans une application un module *Network* pour fournir les fonctionnalités réseaux, un module *Files* pour la gestion des fichiers, un module *GUI* (de l'anglais *Graphical User Interface* pour *Interface Graphique Utilisateur*).

La différence entre un module et une couche est que cette dernière limite totalement les accès aux fonctionnalités de couches juste au dessous et juste en dessous. Un module est plus permissif et autorise les accès à tous les autres modules de la couche. (Mais plus un module à d'interaction avec les autres modules, plus la maintenance du code est complexe.)

Une bibliothèque logicielle peut fournir un module complet, une couche complète, ou seulement certaines fonctionnalités qui devront etre integrees dans vos propres modules ou vos propres couches.

Système d'exploitation

Une couche particulièrement importante est le système d'exploitation. Son rôle est de fournir la couche d'abstraction du matériel (HAL), ainsi que différentes bibliothèques logicielles pour gérer les ressources (mémoire), accéder aux disques et réseau, etc.

Cas particulier des systèmes embarqués
Un système embarqué est un dispositif électronique autonome, qui exécute généralement un programme informatique dédié. Ce type de dispositif ne contient généralement pas de système d'exploitation, ce qui implique que les développeurs doivent écrire du code spécifique à ce matériel. De plus, ce type de dispositif a des contraintes d'utilisation très spécifiques (temps réel, beaucoup plus lent qu'un ordinateur de bureau, avec très peu de mémoire), ce qui impose d'avoir une approche différente pour concevoir des programmes.

Ce cours se focalise sur une approche généraliste du C++, qui n'est pas forcément adaptée pour la programmation embarquée. Il faudra donc adapter les pratiques vues dans ce cours aux contraintes de l'embarqué.

Le cas particulier de la bibliothèque standard du C++

Comparé à d'autres langages de programmation, le C++ est un peu particulier. De nombreux langages, comme le Java, C# ou Python propose des bibliothèques standards assez fournies en termes de fonctionnalités : reseau, graphiques, format de fichiers, etc.

Le C++ a fait le choix de limiter très fortement ce que contient la bibliothèque standard, ce qui implique que les développeurs C++ doivent utiliser d'autres bibliothèques. Certaines bibliothèques sont tellement utilisées qu'elles sont presque de facto des standards, mais vous êtes

libres de ne pas les utiliser si vous le souhaitez (et si vous avez une bonne raison de ne pas les utiliser).

Cette approche très particulière du C++ déroute parfois les développeurs qui ont l'habitude des autres langages. Ils s'attendent à trouver dans la bibliothèque standard tous ce qui permet de créer un programme, sans être obligé d'utiliser des bibliothèques externes (sauf pour des fonctionnalites tres specifiques).

Pour un développeur C++, utiliser une bibliothèque externe fait partie des choses “normales”, ce qui implique un travail de recherche, de comparaison et d'apprentissage des bibliothèques externes.

Ce cours se focalise sur l'apprentissage des bases du C++, ce qui ne permet pas d'étudier les bibliothèques externes. Certaines bibliothèques seront utilisées dans les exercices, mais ca sera a vous d'aller étudier le fonctionnement de celle-ci dans les documentations et tutoriels.
Pour etre plus precis, la bibliothèque standard propose les fonctionnalités suivantes (qui seront étudiées dans la suite de ce cours) :

- les entrées d'informations via un terminal (utilisation du flux d'entrée `std::cin`) ;
- l'affichage de texte dans un terminal (utilisation du flux de sortie `std::cout`) ;
- la lecture et l'écriture de fichiers (utilisation du flux `std::fstream`).

Il faut ajouter les options de commandes, qu'une application reçoit lors de son lancement. Ces informations sont directement traitées par la fonction `main` et font partie du langage C++, pas de la bibliothèque standard.

Ces fonctionnalités sont minimalistes. Il sera par exemple difficile d'utiliser `std::fstream` pour créer un fichier image JPEG ou un système de base de données. De même, il sera difficile de créer une interface

graphique, de gérer la souris ou d'utiliser le réseau.

Il y a deux bibliothèques importantes à connaître :

- boost ;
- Qt.

Framework Application

Les flux (streams)

Par defaut en C++, les IO passent en general par une abstraction : les flux. Deja vu la premiere : `std::cout`. Pour rappel, envoyer des donnees a `std::cout` via l'operateur `«` :

```
std::cout << "un texte";
```

Signifie : on envoie la litterale chaine de caracteres “un texte” au flux `std::cout`. Possibilite de chainer les envois :

```
std::cout << "un texte" << "un autre texte";
```

Avec l'operateur `«`, on parle de flux de sortie. Existe aussi des flux d'entree, pour lire des donnees, avec l'operateur ». L'equivalent de `std::cout` en entree est `std::cin` (pour *C stream input*). Cela correspond au clavier, ca sera vu dans le chapitre “entree console”.

Tampon memoire et flux

Traitement des donnees par un flux = peut etre couteux en performances (temps d'acces aux peripheriques). Pour eviter cela, les donnees ne sont pas traiter tout de suite, mais mise en memoire dans un

tampon (buffer) et traite en paquet (fetch).

Pensez en termes asynchrone entre l'envoi/reception des donnees et le moment ou elle sont traitees.

Note : en cas d'arret brusque de l'application (crash), faire attention que les buffers soient videes si on veut pas perdre les donnees.

Par exemple, `std::endl` fait 2 chose : ajoute un caractere “retour a la ligne” `\n` dans le flux, puis demande de vider le flux (flush). Equivalent :

```
std::cout << std::endl;

// est equivalent a :

std::cout << '\n';
std::cout.flush();
```

Pour des raisons de performances, vous verrez souvent `std::endl` est remplace par '\n' dans les codes, pour eviter le flush. (Dans ce cours, la lisibilite est preferee dans un premier temps aux performances, donc utilisation de “endl” qui est plus explicite).

Autres chapitres

Autre exemple de stream : `fstream` (file stream), qui est une flux d'acces aux fichiers, en lecture et ecriture. Sera vu dans le chapitre “les fichiers”.

Avant cela, un premier chapitre, sans les flux, sur les donnees qui sont envoyees au programme par l'OS lorsque vous lancer un programme (les parametres de ligne de commande).

https://www.owasp.org/index.php/Format_string_attack

Chapitre précédent	Sommaire principal	Chapitre suivant