Programmation mode d'emploi

version 1.0

SI 6 - Développement d’applications

1 - Erreurs de programmation

Même les programmeurs les plus expérimentés font des erreurs. Savoir comment déboguer une application et rechercher ces erreurs est un thème important en programmation. Cependant, avant d'aborder le processus de débogage, il est utile de connaître les différents types de bogues que vous devrez rechercher et corriger.
Les erreurs de programmation se répartissent dans trois catégories : les erreurs de compilation, les erreurs d'exécution et les erreurs de logique. Les techniques de débogage propres à chaque catégorie sont différentes.

1.1 - Erreurs de compilation

Les erreurs de compilation, également connues sous le nom d'erreurs du compilateur, sont des erreurs qui empêchent votre programme de s'exécuter. Lorsque vous compilez votre code dans un langage binaire que l'ordinateur comprend, si le compilateur rencontre un code qu'il ne comprend pas, il génère une erreur du compilateur.
La plupart des erreurs du compilateur sont engendrées par celles que vous faites lors de la saisie du code. Par exemple, vous pouvez mal orthographier un mot clé ou oublier certains éléments de ponctuation nécessaires (parenthèses ou accolades mal appariées, par exemple).

1.2 - Erreurs d'exécution

Les erreurs d'exécution se produisent pendant que votre programme s'exécute. Elles se produisent généralement lorsque votre programme tente une opération impossible.
Par exemple, une division par zéro. Supposez l'instruction suivante :

```
Vitesse = kilometres / Temps
```

Si la variable `Temps` a pour valeur 0, l'opération de division échoue et provoque une erreur d'exécution. Le programme doit s'exécuter pour que cette erreur soit détectée. En revanche, si `Temps` contient une valeur valide, aucune erreur ne se produit.
Dans le cas d'un langage interprété, il peut parfois être délicat de faire la différence entre une erreur de compilation et une erreur d'exécution, sauf en employant un système de gestion des erreurs.

1.3 - Erreurs de logique

Les erreurs de logique empêchent votre programme de faire ce que vous avez prévu qu'il fasse. Votre code peut se compiler et s'exécuter sans erreur, mais une opération peut produire un résultat inattendu.
Par exemple, votre programme peut contenir une variable nommée `Prenom`, dont la valeur initiale est une chaîne vide. Par la suite, votre programme peut concaténer `Prenom` avec une autre variable nommée `Nom` pour afficher un nom complet. Si vous avez oublié d'affecter une valeur à la variable `Prenom`, seul le nom de famille s'affiche, au lieu du nom complet prévu.
Les erreurs de logique sont les plus difficiles à localiser et à corriger. En effet, le programme fonctionne, mais ne donne simplement pas le résultat attendu. Et, dans le pire des cas, cela ne se produit qu'occasionnellement.

2 - Les lois du programmeur

2.1 - Point ne paniqueras

Chaque développeur, lors de son premier cours, a pensé qu'il n'était pas assez bon, pas assez intelligent, que son cerveau ne marche pas comme il faut et qu'il ne comprendra jamais. Sachez juste que n'importe qui peut apprendre à programmer à un niveau basique. Au fur et à mesure, vous allez commencer à comprendre et vous verrez la logique derrière ce charabia. Vous y arriverez d'autant mieux en faisant abstraction du langage, et en apprivoisant les concepts sous-jacents à tous les langages : les conditions, les comparaisons, les types de variables… bref, l'algorithmique.
Pour devenir bon, il n'y a pas de recette miracle : juste du travail, et encore du travail (comme partout hein !). Ecrivez des petits programmes basiques, cherchez sur Internet, lisez des articles. La patience est le maître-mot pour devenir un bon codeur !

2.2 - Le code tu formateras

Il est très facile de reconnaître un bout de code écrit par un débutant, uniquement d'après la mise en forme du code. Une indentation inégale, des retours à la ligne impromptus, des ouvertures d'accolades toujours différentes : ce sont là les "erreurs" que vous devez apprendre à éviter. Un code proprement formaté est beaucoup plus lisible. Une indentation bien faite peut vous éviter des heures à chercher pourquoi votre code ne marche pas, parce que vous avez oublié de fermer une accolade dans tout un tas de boucles imbriquées.
Les différents langages possèdent des standards de formatage, certains langages étant extrêmement permissifs quand d'autres sont intransigeants. Parfois, une entreprise impose un style de formatage donné. Mieux vaut prendre les bonnes habitudes dès le début !

2.3 - Attention à la casse tu feras

Certains langages sont sensibles à la casse (ils différencient majuscules et minuscules, par exemple Javascript), et d'autres non (par exemple HTML). Mais peu importe le langage que vous utilisez, vous devez vous fixer une règle sur l'utilisation de majuscule et minuscule, et vous y tenir rigoureusement.
Il est fréquent de déclarer une variable avec une majuscule (comme par exemple "Noob = true") pour y faire référence plus tard avec une minuscule (du genre, "if(noob)"). C'est la meilleure façon de perdre du temps à comprendre pourquoi le code ne marche pas.
Chaque langage a ses conventions de nommage et on a souvent tendance à importer une convention quand on change de langage. Mieux vaut consacrer quelques instants à vérifier les règles à appliquer. Cela sera autant de temps économisé par la suite.

2.4 - Aux variables et fonctions un nom adéquat tu donneras

Les noms rencontrés dans certains langages peuvent faire sourire (comme `AbstractSingletonProxyFactoryBean` en Java), mais on en comprend vite l'intérêt. Déclarez toujours des noms de variables ou de fonctions les plus descriptifs possibles, pour éviter d'avoir des noms identiques dans votre code. Il reste fréquent de déclarer une variable compteur (i, par exemple) et de re-déclarer plus loin la même variable. En cas de boucles imbriquées, si vous n'avez pas fait attention à l'initialisation, c'est source d'ennuis. Il est fréquent de privilégier la rapidité et de déclarer des variables le plus simplement possible (juste une lettre).

À la relecture toutefois, vous passez trois fois plus de temps à devoir redéterminer à quoi correspondent « `a` », « `nb` » ou « `toto` ».
Un dernier point : attention aux fautes d'orthographe sur les noms de variables ou de fonctions. C'est déjà assez compliqué de se remémorer de toutes les variables, alors si en plus vous devez vous rappelez de la faute....

2.5.- A bon escient mais sans excès tu commenteras

Une des premières choses que vous allez apprendre en commençant à programmer, c'est que vous **devez** commenter votre code. Un code commenté proprement (c'est à dire qui décrit ce que font vos fonctions, qui décrit l'algorithme que vous avez mis en place, etc ..) permettra à celui qui reprendra votre code (ou à vous même) de mieux comprendre le pourquoi du comment.
Il faut toutefois éviter de tomber dans l'excès et de commenter chaque ligne de son code. L'exemple suivant n'est pas si rare :

```
score += 5; // Ajoute 5 au score
```

Commentez intelligemment. Dire que vous ajoutez tel nombre à telle variable est complètement inutile. En revanche, décrire le pourquoi et le comment de votre fonction est un des réflexes que vous devez acquérir au plus vite.

2.6 - Des outils de débogage tu te serviras

Tôt ou tard, vous vous retrouverez face à un programme qui semble marcher mais qui ne fait pas ce que vous souhaitez. C'est à ce moment là que les outils de débogage seront vos meilleurs amis.
Certains IDE (environnement de programmation) mettent à votre disposition des outils permettant de voir, pas à pas, ce que fait votre programme, par où il passe, les valeurs de vos variables à l'instant T … Abusez-en.

2.7 - Aux sauvegardes et versionning tu penseras

Cela ne concerne pas uniquement la programmation, mais personne n'est à l'abri d'une coupure de courant, d'une défaillance disque ou de code perdu lorsque vous travaillez en groupe (parce que votre collègue, qui travaillait sur le même fichier que vous, a tout écrasé). Il existe tellement d'outils de versionning et de sauvegarde que vous ne **devez** pas perdre de code.

2.8 - Humilité tu conserveras

Votre programme marche comme il faut, et vous en êtes naturellement fier ! Les suivants aussi : votre confiance grandit, vous pensez tout savoir. Mais survient alors un autre projet où vous piétinez dès le début. Le doute s'instaure, et vous abandonnez.
Restez donc humble : le monde de l'informatique est en constante évolution. Chaque jour, vous aurez de nouvelles choses à apprendre. C'est là la beauté et la malédiction de notre profession.
Lisez tout, informez-vous, discutez avec d'autres. Testez les nouveaux outils et les nouvelles charpentes. .

2.9 - Conclusion

Le maître-mot du développeur est la réflexion. Souvenez-vous bien qu'un code n'est pas **la** solution, mais une expression d'une solution. Alors seulement serez-vous prêt à devenir un bon codeur.

3 - Les erreurs de programmation les plus dangereuses

La CWE (*Common Weakness Enumeration*) élaborée chaque année par MITRE répertorie les erreurs de programmation les plus dangereuses. Si toutes ces vulnérabilités ne s'appliquent pas forcément au développement web, elles sont tout de même intéressantes à étudier, ne serait-ce que pour se former aux bonnes pratiques de développement.

3.1 - Mauvaises interactions entre les composants

Ces vulnérabilités concernent les manières dont les données sont envoyées et reçues par des composants, modules, programmes, processus, threads ou systèmes.

- Mauvais filtrage des données envoyées par les utilisateurs.
- Mauvais encodage ou échappement des données en sortie.
- Mauvais nettoyage d'éléments spéciaux utilisés pour une **commande SQL** (injection SQL)
- Non-conservation de la structure des pages Web (utilisation de scripts intersites *Cross Site Scripting* (ou XSS).
- Mauvais nettoyage d'éléments spéciaux utilisés dans une commande d'OS ("*OS Command Injection*", par system() ou exec()).
- Falsification de demande intersite (*Cross-site request forgery*, **CSRF**).
- Situation de concurrence.
- Divulgation d'informations dans les messages d'erreur (comme le PATH dans lequel est installée votre application web).

3.2 - Mauvaise gestion des ressources

Ces vulnérabilités concernent la manière dont les applications ne gèrent pas correctement la création, l'utilisation, le transfert et la destruction des ressources systèmes.

- Copie dans le tampon sans contrôle de la taille de l'entrée ("***Classic Buffer Overflow***") .
- Modifications externes de données d'état.
- Modification externe des chemins d'accès ou des fichiers en lecture / écriture.
- Limitation erronée d'un chemin d'accès à un répertoire restreint ("*Path Traversal*")
- Injections de code (eval()).
- Téléchargement et exécution de code sans réel contrôle d'intégrité.
- Mauvaise libération de ressources (*double free()*…).
- Mauvaise initialisation.
- Erreurs de calcul (off by one, division par zéro).
- Affectation de ressources sans limites

3.3 - Protections foireuses

Il s'agit de modes de protection souvent mal utilisés, contournés, ou purement ignorés par les développeurs.

- Mauvais contrôle d'accès (autorisation) ou absence d'ACL.

- Utilisation d'un **algorithme de chiffrement** décrypté ou dangereux.
- Utilisation d'informations d'authentification **codées en dur**.
- Octroi erroné d'autorisations pour les ressources critiques
- Exécution de programmes avec des permissions trop élevées (le bonheur des CGI en root).
- Prise des mesures de sécurité côté client.