Les étapes et les outils de développement des programmes

L'élaboration d'un programme est un travail qui se fait en plusieurs étapes nécessitant divers outils que ce chapitre tente de décrire succinctement.

La conception

L'analyse

La première étape, probablement la plus intéressante, l'analyse, est une opération purement intellectuelle. Partant d'un cahier des charges (dans le meilleur des cas) cette étude consiste à décrire ce que doit réaliser le programme sans tenir compte ni des spécificités de la machine sur lequel il devra tourner ni du langage dans lequel il sera écrit.

La traduction dans un langage de programmation

L'algorithme produit par l'analyse est traduit dans un texte appelé *code source*. Celui-ci est écrit dans un langage choisi par le programmeur ou imposé par les usages en vigueur dans son entreprise en fonction du projet dans lequel devra s'intégrer son programme, en fonction aussi des habitudes de ses collègues ou encore des logiciels dont dispose l'entreprise pour développer des applications semblables.

Le langage choisi devra pouvoir exprimer parfaitement toutes les subtilités de l'algorithme issu de l'analyse.

Un *langage de bas niveau* tel que le *langage assembleur* pourrait être envisagé pour des programmes devant interagir directement avec le matériel, des pilotes de périphériques ou des systèmes embarqués par exemple. On limitera cependant l'usage de l'assembleur à quelques bouts de codes agissant directement sur le matériel ou pour la programmation de processeurs hyper spécialisés tel que les DSP (*Digital Signal Processor*)

Les *langages de haut niveau* ont été créés pour accélérer l'écriture des programmes. Les programmeurs peuvent grâce aux *langages évolués* faire abstraction de l'architecture de l'ordinateur et du processeur et se consacrer uniquement à la logique de leurs algorithmes.

Le C, le C++ ou le C# donnent presque autant de possibilités que l'assembleur et sûrement autant que le Pascal et son successeur orienté objet le Delphi pour les applications Windows ou Kylix pour les applications sous linux.

Le Visual Basic successeur Microsoft du Basic permet de créer facilement de petites applications Windows qui comme pour le Delphi peuvent très facilement s'interfacer avec des bases de données.

Le Java conçu par les ingénieurs de SUN est un langage dont la syntaxe est proche du C++. Le but des concepteurs du Java était d'établir un langage indépendant des machines. Il est destiné à être traduit non pas en langage machine mais en un langage intermédiaire appelé *byte code*. Ce dernier est portable sur n'importe quelle machine pour peu qu'un programme approprié, une *machine virtuelle*, y soit installé pour interpréter ce byte code.

Le C# est la réponse de Microsoft, dans la lignée du C++ en reprenant des caractéristiques du Java il facilite la création d'applications Windows ou pour le Web.

Au sujet du Web, nous avons vu naître ces dernières années un certain nombre de langages conçus spécifiquement pour ce genre d'application : PHP, Perl, etc. Ce sont des langages d'écriture de *scripts* tout comme le JavaScript ou le VBScript aussi utilisé pour créer des lignes de codes intégrables aux applications réalisées avec Microsoft Office.

La production du programme

1 L'édition du code source

Les algorithmes une fois traduits dans un ou des langages de programmations sont des textes à encoder à l'aide d'un *éditeur de texte*.

Les logiciels de traitement de texte apportent des artifices de mise en page et de style tout à fait inutiles pour cette tâche. Les éditeurs de textes tels que Edit ou le bloc note de Windows feront bien mieux l'affaire. Des éditeurs spécialisés offrent la possibilité d'avoir une syntaxe colorée qui selon le langage de programmation distingue les mots clés, labels, mots réservés, signes de ponctuation, délimiteurs, nombres, chaînes de caractères etc. en attribuant à chacun de ces éléments des couleurs distinctes ce qui en facilite la lecture.

Mieux encore, on dispose souvent d'Environnements de développement Intégrés (IDE *Integrated Development Environment*) qui regroupent dans une même application l'éditeur de texte, des éditeurs graphiques pour les formulaires, les boîtes de dialogue et les images, ainsi que d'autres outils indispensables à la construction d'une application : le compilateur, l'éditeur de liens, le débogueur, un système de gestion des versions etc.

2 La traduction en langage machine

Le code source rédigé dans un langage adapté à notre esprit humain doit maintenant être traduit en un autre langage compréhensible par la machine. Cette conversion peut être faite de deux manières : l'interprétation ou la compilation.[1]

2a L'interprétation

Un interpréteur est facile à utiliser, l'exécution du programme est faite sans transition immédiatement après l'édition. L'interpréteur analyse le code source et exécute de suite les instructions machine qui correspondent à chaque ligne lue. La vitesse d'exécution s'en ressent puisque chaque ligne du programme doit à chaque fois être interprétée avant d'être exécutée.

2b La compilation

La compilation est une traduction du *code source* en langage machine faite une fois pour toutes. Le code produit par le compilateur est appelé *code objet*. C'est un code binaire qui nécessite encore quelques manipulations pour être exécutable mais dès à présent le code source peut être mis de côté. Il n'est plus utile pour l'exécution proprement dite de l'application. Seuls les programmes de mise au point pourraient encore y recourir pour aider les développeurs à visualiser les instructions lors de leur exécution.

La compilation se fait par étapes successives que nous décrivons brièvement car cela nous aide à comprendre la structure des langages de programmation.

- L'analyse lexicographique

C'est la découpe des lignes de programmes en symboles (labels, mots réservés, constantes numériques, opérateurs logiques ou mathématiques, signes divers …). Les espaces et les commentaires utiles pour la présentation du code source peuvent dès à présent être ignorés.

---

[1] Par rapport à ces deux méthodes de traduction, le Java occupe une position intermédiaire puisqu'il est d'abord compilé pour produire le byte code puis ce byte code est interprété par la machine virtuelle Java.

- L’analyse syntaxique

Les symboles résultant de l’analyse lexicale ont chacun une signification, on parle d’unités syntaxiques. Ces symboles doivent aussi être agencés de manière convenable pour construire des lignes de code sensées. Le programmeur a dû respecter des règles de syntaxe, une "grammaire", propres au langage choisi. L’analyse syntaxique vérifie que le code qu’il a écrit est conforme aux règles d’écriture en usage. S’il y a des erreurs ou des ambiguïtés à ce point de vue, le compilateur sera incapable de poursuivre la traduction du code source en langage machine.

Exemples de règles de syntaxe ou de grammaire :

- En langage naturel, une phrase se compose généralement d’un sujet, d’un verbe et d’un complément.
- En langage C, une condition s’exprime par le mot clé "if" suivi d’une expression entre parenthèses qui exprime la condition puis d’une ou de plusieurs instructions mises entre accolades.
- Dans une expression, le nombre de parenthèses ouvrantes est égal au nombre de parenthèses fermantes.

- L’analyse sémantique

Les questions de "vocabulaire" et de "grammaire" étant réglées, reste à découvrir le sens de ce que le programmeur a voulu écrire.

- Quels sont les objets traités par le programme ? Ils sont désignés par des identificateurs qui correspondent en pratique à des adresses.
- Quelles sont les propriétés de ces objets ? Ceux-ci sont parfois "déclarés" au préalable. Cette déclaration obligatoire dans les langages les plus rigoureux est parfois facultative ou implicite avec d’autres langages moins contraignants.
- Le type de l’objet doit être connu pour savoir comment interpréter les opérations qui lui sont associées. Un signe ‘+’ aurait un effet différent s’il concerne des nombres entiers, des nombres réels ou des chaînes de caractère.
- La taille de l’objet a une incidence sur l’espace qu’il occupera en mémoire
- La durée de vie de l’objet doit être connue du compilateur pour qu’il puisse décider du type d’emplacement qui lui sera alloué (dans un registre, sur la pile, en mémoire...) Faudra-t-il prévoir des instructions pour créer et détruire cet objet ?

- La génération du code et son optimisation

Cette dernière étape consiste à produire le code binaire qui exécutera les actions équivalentes à celles qui sont demandés dans le code source. Il n’est plus question ici de s’abstraire des particularités de la machine sur laquelle devra tourner l’application. Le code sera optimisé parfaitement si le compilateur parvient à produire un code aussi efficace que si le programme avait été écrit directement en langage d’assemblage.

Remarques à propos des « compilateurs »

1. Un assembleur est un compilateur un peu particulier puisqu’il traite un code source qui n’est pas un langage évolué. Sa tâche, notamment en ce qui concerne la génération de code est beaucoup plus simple puisqu’il y a une correspondance directe entre les mnémoniques et les codes opérations à produire.

2. Un compilateur croisé (*Cross compilator*) est un compilateur qui produit des codes machines pour des machines cibles ayant un processeur d'une famille différente de celui de la machine de développement. Exemple : Un compilateur qui tourne dans un PC, et qui est donc un programme fait d'instructions pour microprocesseurs Intel, mais qui produit du code pour des microprocesseurs Motorola.

3 L'édition de liens

Les programmes dès qu'ils deviennent d'une certaine importance ne peuvent plus être écrits d'un seul bloc. Il est préférable de les découper en modules pouvant être pensés, édités, modifiés, compilés et idéalement testés séparément. Ces modules ne doivent pas nécessairement être écrits dans un même langage. La découpe d'une application en module est laissée à l'appréciation de son concepteur. Cette réflexion doit être faite au moment de l'analyse.

Les codes sources, appelés modules sources, une fois compilés donnent ce qu'on appelle des modules objets. Ces modules sont divisés en segments, généralement un segment de code, un pour les données, un pour la pile etc.

Les modules doivent maintenant être rassemblés pour former le code exécutable. Cette tâche est confiée à l'éditeur de liens, le « *linker* ».

Les compilateurs ne connaissent pas les emplacements exacts des variables ni des fonctions qui constituent les modules. Ils ne peuvent déterminer que les emplacements relatifs au module lui-même ou plus exactement aux segments qui constituent le module.

Les modules objets sont donc relogeables, on dit « relocatables » Cela signifie que les adresses des variables et des fonctions vont être ajustées en fonctions des tailles et des positions relatives des modules qui vont être combinés. Les modules sont rassemblés en regroupant les segments de même nature (code, données, pile, …) Après édition de liens, les adresses du code exécutables sont relatives aux débuts des segments. A ce stade, l'exécutable est lui aussi « relocatable »

C'est au moment de l'édition de lien que se font les jonctions entre données ou fonctions externes (liens à satisfaire) et les données et fonctions qui ont été déclarées publiques (lien utilisables). Certaines de ces fonctions proviennent de bibliothèques (*library*) qui ne sont autres que des collections de modules objets contenant des fonctions déjà développées et testées séparément telles que des fonctions de calculs ou celles qui servent aux interfaces utilisateur.

L'éditeur de lien ne peut accomplir sa tâche que si tous les liens peuvent être résolus, seule exception : les liens dynamiques, ils ne seront résolus qu'au moment de l'exécution du programme.

◊ *La relocation*

Cette opération réalisée par un programme appelé « *locator* » n'est faite que pour les applications « *stand alone* » dont le code est destiné à être programmée dans une mémoire morte. Les directives données au locator lui indiquent l'emplacement exact où devront se trouver le code et les données.

4 Le chargement

Le plus souvent la version exécutable est chargée en mémoire par le chargeur ou « *loader* » un sous-ensemble du système d'exploitation qui positionne l'exécutable en mémoire, ajuste les adresses en fonction de l'emplacement exact du programme et lance son exécution.

5 Le Debug

Le debug ou déboguage est une étape essentielle dans le processus d'élaboration d'un programme. Cette phase est trop souvent négligée dans l'apprentissage de la programmation, comme s'il était possible de faire des programmes sans erreurs ! On a beau être rigoureux dans l'écriture de son programme, l'avoir préparé le mieux possible dans un cahier des charges, avoir lu et relu son code… il faut toujours tester le résultat et presque toujours … on découvre des bugs. Commence alors une véritable enquête, pourquoi cette erreur se produit-elle ? On tente alors des hypothèses, l'erreur se produit dans telle ou telle circonstance. Vérification, l'hypothèse était valable ou pas, poursuite de l'enquête … un travail qui demande beaucoup de réflexion et qui vous apprend à remettre en question votre façon de commenter vos programmes, celle de nommer vos variables, d'organiser vos projets en modules etc.

Les outils de debug sont souvent performants, ils font généralement partie d'environnements de développement intégrés (IDE). Apprendre à les utiliser nécessite un certain temps. C'est une dernière étape dans l'apprentissage de la programmation.

Sans débogueur approprié on est réduit à devoir inséré ça et là dans le programme des instructions du style « coucou je suis dans la fonction machin » ou encore « la variable = *xyz* »

Un débogueur vous permet d'exécuter votre programme pas à pas, des points d'arrêt placés aux endroits critiques vous laissent le temps de consulter les variables et même de les modifier pour voir comment va réagir le programme. Une trace ou l'état de la pile vous aident à comprendre ce qui s'est passé avant d'aboutir au point d'arrêt. Le débogueur vous permet donc de découvrir que sans doute votre programme s'est déroulé autrement que ce que vous ne l'aviez prévu et pourtant exactement (bêtement) comme vous l'aviez écrit !

Table des matières