CHAPITRE 1

PRINCIPES DE LA PROGRAMMATION

Présentation

Avant toutes choses il est bon de rappeler les définitions de deux mots clés souvent utilisés tout au long de ce livre.

L'informatique est la science du traitement automatique et rationnel de l'information mettant en cause des matériels (ordinateurs) et des logiciels.

L'ordinateur est une machine automatique de traitement de l'information obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.

En clair, l'informatique est une discipline qui permet de classer, stocker, et traiter tous types de données appelées "informations" il peut s'agir de données commerciales, scientifiques, comptables, d'images, textes, vidéos ou autres. Les machines qui permettent ces traitements de données sont les ordinateurs, leur fonctionnement est très souvent automatisé grâce à des programmes ou logiciels qui permettent l'exécution de tâches bien définies. Un programme exécute des fonctions bien précises. Des erreurs ou des fantaisies lors de son exécution ne proviennent pas d'une initiative de l'ordinateur mais d'un programme présentant une erreur de conception.

Langage de bas niveau et langage machine

L'appellation langage de bas niveau signifie langage proche de la machine et donc de son processeur. Il s'agit de langages utilisant le binaire ou des instructions restreintes quasi incompréhensibles et difficiles à manier pour un individu quelconque. L'ordinateur ne peut lire que des codes binaires (succession de 0 et de 1) définis par le constructeur pour réaliser des instructions de base comme les transferts de données entre la mémoire centrale et l'unité de traitement, les instructions de calcul (arithmétique, logique ...).

Au tout début l'écriture des programmes était réalisée en langage machine en mettant bout à bout des 0 et des 1. Tout cela était extrêmement long et pénible, les erreurs étaient fréquentes. Puis ont été utilisés des codes mnémoniques (LOAD, MOVE, JUMP, ADD, DIV...) plus faciles à manier. La traduction en langage machine est alors assurée par un langage d'assemblage plus généralement appelé assembleur. Il permet d'exploiter au mieux les ressources de la machine mais sa mise en œuvre n'est pas facile. D'où la naissance des langages de programmation structurés ou langages de haut niveau.

Langage de haut niveau

Les langages évolués ou dits de haut niveau sont plus compréhensibles par l'homme ; ils comportent des instructions bien plus lisibles comme :

```
for i in 1..10 loop
put (" bonjour");
end loop;
```

L'ensemble de ces lignes de programme constituent le **code source**.

Un programme appelé **interpréteur** ou **compilateur** est ensuite chargé de transformer ces instructions en un code machine exécutable par le microprocesseur.

Un langage de programmation est défini par des règles d'écriture que devront respecter les programmes pour être traduits en binaire. Chaque langage de programmation impose des mots clés comme *: begin, integer, procedure, end, if then else, return...*, et des règles de construction qui correspondent à la forme particulière que devront avoir les programmes. La difficulté, pour le programmeur ou développeur, consiste à respecter ces règles imposées, qui, de plus, varient d'un langage à l'autre. Il faut donc apprendre un langage de programmation à l'instar d'une langue étrangère. Nous verrons plus tard que la bonne connaissance d'un langage tel qu'ADA ou PASCAL permet de rapidement programmer en C, C++ ou JAVA, les règles de base étant très voisines pour bon nombre d'instructions.

Pour des programmes plus élaborés et complexes les développeurs et chefs de projets, avant d'éditer le code source, écrivent des algorithmes.

Un algorithme est l'écriture dans un langage clair et littéral, pour un individu, de la méthode informatique de résolution d'un problème. Le programmeur devra ensuite traduire son algorithme dans le langage de

programmation qu'il souhaite utiliser et dans le respect des règles imposées par le cahier des charges.

Il existe deux grandes sortes de langages de haut niveau :
les langages interprétés comme : Basic, Lisp, Logo, Prolog, et la plupart des langages à script.

les langages compilés tels Ada, Fortran, Pascal, C, C++, C#, etc.

Le programme source
L'ensemble des lignes de programme écrites par le développeur (ou programmeur) constituent le code source. Elles sont écrites à l'aide d'un éditeur de texte. Il peut ainsi modifier une instruction, en taper d'autres comme avec un traitement de texte.

Le programme exécutable
Le programme exécutable est constitué de 0 et de 1 exploitables par le processeur de l'ordinateur. Il est généré à partir du programme source.

Programmes interprétés
Pour un langage interprété, le programme source va être lu, ligne après ligne, par un programme spécifique appelé interpréteur qui, pour chaque ligne rencontrée, va analyser l'opération à effectuer, la vérifier puis la traduire en langage machine et l'exécuter immédiatement via le processeur de l'ordinateur.

L'interpréteur répète donc l'ensemble des trois instructions suivantes autant de fois qu'il rencontre une ligne de programmation :

Lecture d'une ligne du code source ;
Analyse, vérification et codage binaire de la ligne ;
Exécution de l'instruction associée à la ligne interprétée ;

L'interpréteur présente des avantages comme la disponibilité du programme source que l'on pourra donc modifier, la mise au point rapide de petits programmes ainsi que leur exécution. Cependant deux inconvénients majeurs existent : il n'existe pas de programme exécutable et l'interpréteur doit être donné avec le programme source si l'on veut exécuter ce dernier. Enfin l'interpréteur lorsqu'il doit repasser sur une ligne de code la retraduit à nouveau d'où la lenteur d'un programme interprété par rapport à un programme compilé.
Par exemple, si le programme tombe dans une boucle (voir chapitre 4) où doivent être répétées 1000 fois 3 instructions A, B et C ; au lieu de transcrire en binaire une seule fois chacune de ces trois instructions, l'interpréteur

transcrira 1000 fois chacune d'elles soient 3000 transcriptions au lieu de 3. Tout cela prend du temps et aura des conséquences pour l'utilisateur.

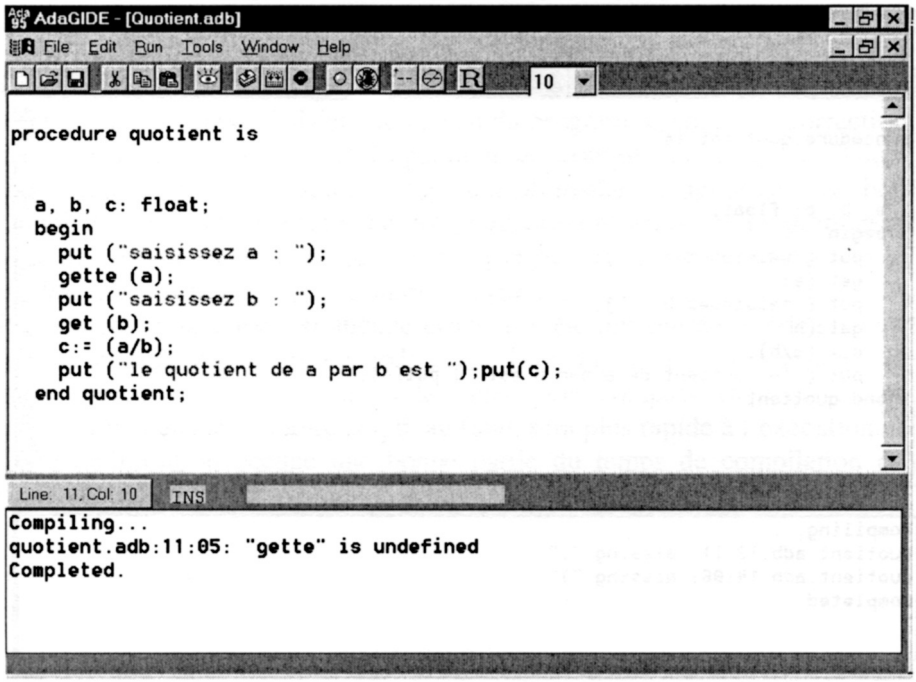
Programmes compilés
Pour un langage compilé, le programme source écrit par le programmeur va être optimisé et traduit dans son ensemble en un fichier binaire exécutable par le microprocesseur de l'ordinateur.
Une fois compilé, le programme n'a plus besoin d'accéder au code source et le compilateur n'est plus nécessaire contrairement à un programme interprétable.
Un programme exécutable (fichier binaire) est généré une fois pour toutes. L'exécutable est cependant dépendant du microprocesseur pour lequel il a été compilé. Cela signifie qu'il ne fonctionne pas nécessairement sur des ordinateurs différents.
Le développement d'un programme compilé est plus lourd que pour un programme interprété (répétition fréquente des quatre étapes suivantes : édition, compilation, édition de liens, tests) mais l'exécution est bien plus rapide.

Fonctionnement d'un compilateur

Une fois finie l'écriture du programme source, le développeur informatique lance l'opération de compilation. Il s'agit d'utiliser un programme "le compilateur" qui relit les lignes de code et recherche les éventuelles erreurs lexicales, syntaxiques ou sémantiques, optimise le programme et délivre un programme exécutable par la machine.

Analyse lexicale
Durant cette première phase, le compilateur parcourt l'ensemble des chaînes de caractères pour produire des éléments du langage à l'image d'un lecteur qui associe des lettres pour former des mots. Le compilateur met de côté les éventuels commentaires du programme, inutiles pour la compilation.
A ce niveau sont détectées les erreurs telles : des variables trop longues, des caractères, des nombres ou mots inconnus par le langage de programmation.

Dans la fenêtre ci-après, le compilateur ADA signale au développeur que l'instruction *gette* n'est pas définie : c'est-à-dire qu'elle n'existe pas dans le langage. Pour mémoriser le contenu d'une variable l'instruction ADA est *get*. Il y a donc une erreur à la compilation qu'il faudra corriger avant de relancer une nouvelle passe de compilation.



Analyse syntaxique

La syntaxe est l'ensemble des règles d'écriture des phrases d'un programme permises dans un langage de programmation et formant la grammaire de ce langage. Durant cette phase, le compilateur vérifie si le texte constituant le programme source est conforme à la syntaxe du langage de programmation utilisé. Par exemple, toutes les erreurs concernant l'utilisation de mots réservés pour des noms de variables, les opérateurs, les signes de ponctuations, accolades, parenthèses, point-virgules de fin de ligne, les délimiteurs de procédures et fonctions, la syntaxe des instructions, sont signalées.

Exemple :

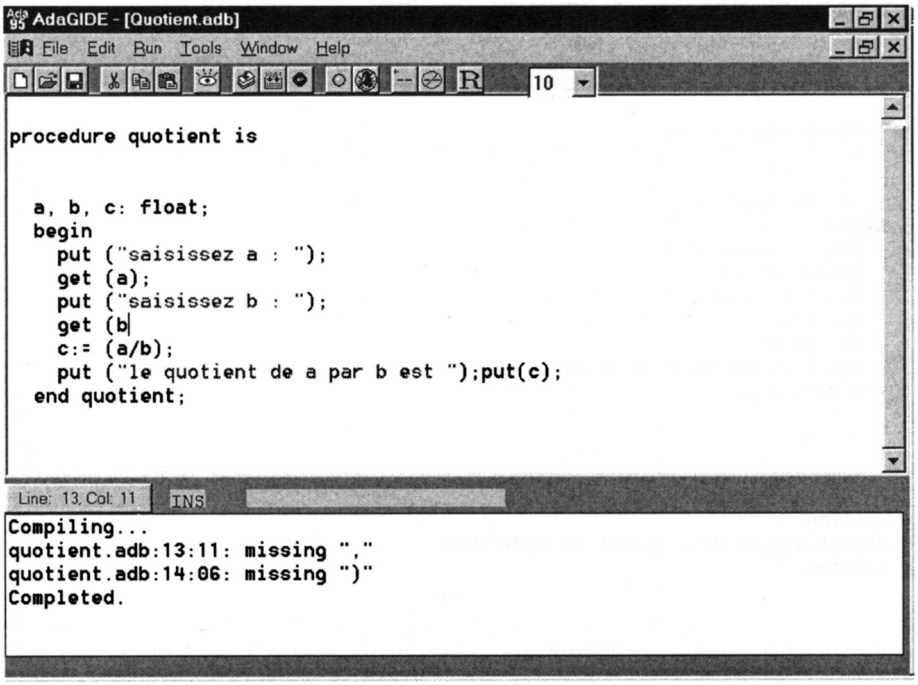
Pour les deux lignes du programme ci dessous :

if a>b then
put ("le plus grand est ") ; put(a);

Le compilateur signalera ici une erreur de syntaxe le point après la parenthèse doit être remplacé par un point-virgule.

La bonne rédaction doit être :

if a>b then
put ("le plus grand est ") ; put(a);



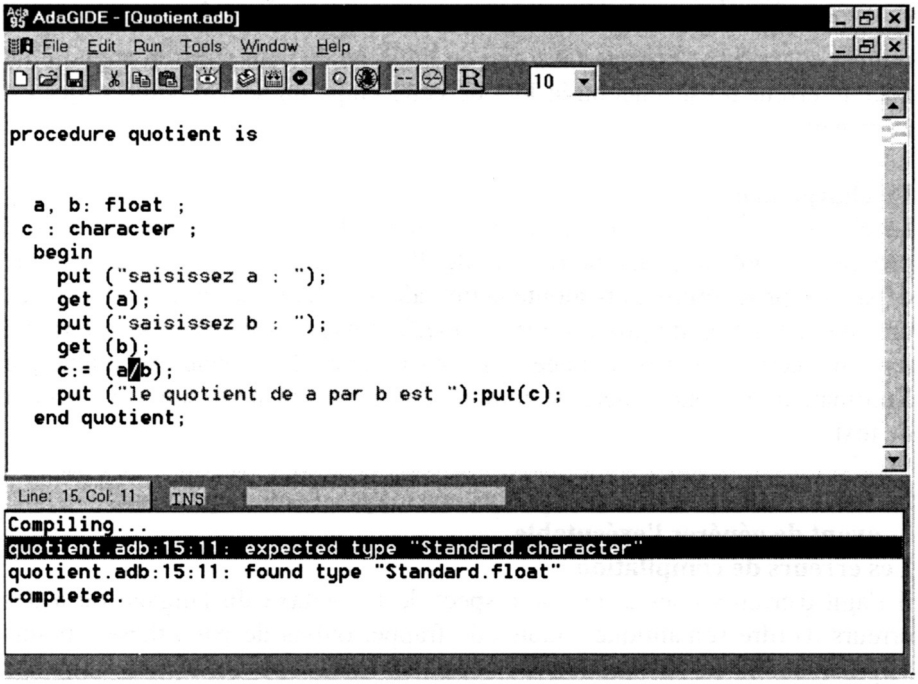
Ci-dessus, le compilateur ADA signale au développeur qu'il manque un signe de ponctuation pour continuer la ligne ou la finir et éventuellement une parenthèse fermante. Pour mémoriser le contenu d'une variable l'instruction ADA syntaxiquement correcte est *get (b);* une erreur est signalée lors de la compilation qu'il faudra corriger avant de relancer une nouvelle passe de compilation.

Analyse sémantique

La sémantique est l'étude de l'ensemble des propositions d'une théorie déductive du point de vue de leur vérité ou de leur fausseté. En clair, cela signifie qu'à ce niveau, le compilateur vérifie, entre autres, la correspondance des types afin que les opérateurs (opérations) agissent bien sur des opérandes (constantes ou variables) adaptés. Le compilateur vérifie par exemple que les bons nombres sont utilisés avec les bonnes opérations. La multiplication de deux variables contenant des chaînes de caractères sera signalée comme une erreur (incompatibilité entre l'opérateur "*" et des opérandes ayant le type "chaîne de caractères" car multiplier des chaînes de caractères n'a pas de sens pour le compilateur ; la bonne déclaration des variables et des procédures et fonctions est également vérifiée lors de cette étape.

Ci-après, le compilateur ADA signale au développeur que l'instruction *c := a/b;* présente un problème (de sémantique). En effet c est une variable de type *character* dans laquelle on pourra stocker un caractère et a/b est de type

nombre réel ou *float* : il y a incompatibilité du typage. Nous reviendrons plus tard sur les types des variables. Cette erreur au niveau du programme source devra être corrigée avant de relancer une nouvelle passe de compilation.



Avertissements ou "warning"

En plus des messages d'erreurs, le compilateur peut délivrer des avertissements ou *warning* en anglais. Ceux-ci n'empêchent pas la délivrance d'un code objet mais attirent l'attention du développeur vers des erreurs qui pourraient intervenir par la suite telles la déclaration d'une variable ou d'une bibliothèque que l'on n'utilise pas.

La génération du code objet

Tant que des erreurs subsistent au niveau du programme source la correction de celles-ci est inévitable et l'opération de compilation relancée. Toute modification du code source oblige une nouvelle compilation. Un bon compilateur décèle les erreurs de programmation et avertit le programmeur avec des messages écrits d'indication (exemples *: missing ";" at the end of the line, declaration is missing, name is expected*).

Passées ces trois étapes, si aucune erreur n'a été rencontrée le compilateur génère un code provisoire dit code objet. Le compilateur optimise aussi le code source en générant un code objet plus compact et donc moins encombrant dans la mémoire et qui, au final, sera plus rapide à l'exécution.

Cette optimisation occupe une bonne partie du temps de compilation du programme source, les instructions redondantes sont également supprimées.

L'édition de liens
Pour d'importants programmes décomposés en plusieurs paquets ayant chacun une tâche bien définie, le compilateur génère autant de fichiers binaires objets que de fichiers sources. Dans un second temps, les liens entre les différents paquets, fonctions, procédures, librairies ou bibliothèques utilisés sont pris en compte : il s'agit de l'édition de liens (*link* en anglais). Si aucune erreur n'est rencontrée lors de cette étape un seul module **objet** est alors généré.

Le chargement
Le chargeur, habituellement couplé à l'éditeur de liens, s'occupe de loger le programme obtenu en mémoire centrale. Il repère donc l'adresse mémoire de départ du programme puis ajoute cette valeur à l'adresse de chargement de chaque instruction du programme et modifie toutes les adresses se référant à une instruction ou une donnée. Un programme réellement exécutable par l'ordinateur est donc généré, le compilateur n'est plus nécessaire ni l'éditeur de texte.

En résumé nous pouvons dire qu'il y a trois types d'erreurs possibles :
1- avant de générer l'exécutable
Les erreurs de compilation :
il s'agit d'erreurs dues à un non-respect de la syntaxe du langage ou à des erreurs d'ordre sémantique : fautes de frappe, oublis de parenthèses, points virgule, oublis de déclaration de variables, mauvaise cohérence entre les données, structures erronées...
Cependant, une fois le module exécutable généré, le programme peut ne pas bien fonctionner.

2- après avoir généré l'exécutable
Les erreurs de logique :
le programme s'exécute mais ne donne pas les bons résultats il peut s'agir de tests conditionnels mal conçus, de variables possédant des contenus erronés ou non prévus, plus généralement d'une mauvaise conception du programme.

Les erreurs à l'exécution :
L'accès à des données introuvables, la demande d'exécution d'une opération mathématique non réalisable entraîne l'arrêt du programme par le système d'exploitation.

Le débogueur

Les erreurs de logique et d'exécution ne sont pas détectables à la compilation mais visibles uniquement lors du mauvais fonctionnement du programme