ANNEE UNIVERSITAIRE 2002 – 2003
COURS : ALGORITHMIE ET PROGRAMMATION
AUTEUR : FABRICE GONTARD

NOTIONS DE BASE SUR L'ALGORITHME

1. INTRODUCTION

L'une des plus grandes difficultés lorsqu'on commence à étudier la programmation est de s'obliger à mettre ses idées sur papier. Quel intérêt ? Pour quoi faire ? Qui ne s'est jamais posé ces questions pour justifier le fait qu'il passe directement sur l'ordinateur pour composer son œuvre ? Or nous pouvons observer autour de nous que dans n'importe quel corps de métier, pour être clair ou pour mieux structurer une idée, les gens griffonnent des schémas sur un papier, sur le coin d'une table ou d'un bureau. En programmation, il doit être de même afin d'obtenir un résultat de qualité.

En fait, un ordinateur ne fait que ce qu'on lui demande de faire. Il est incapable de prendre du recul par rapport à un problème afin d'en optimiser la solution. C'est donc au programmeur qu'incombe cette tâche. Seulement, pour les programmeurs débutants voire expérimentés, il existe beaucoup d'écueils à éviter lors de la conception d'une nouvelle application. En général :

- Persuadé d'avoir une bonne idée, le programmeur cherche à écrire un programme dans un langage donné, pour une machine donnée, avant même d'avoir complètement spécifié le schéma de résolution du problème en cours d'étude ;
- Fasciné par les particularités du langage utilisé, le programmeur dévie progressivement des objectifs de l'application en cours. Il modifie volontairement ou involontairement l'énoncé du problème pour se plier aux contraintes du langage (et non l'inverse) ou se rapprocher des clichés qui l'attirent (découvrir le nouvelles possibilités du langage utilisé=. En bref, par facilité, il se laisse guider par les contraintes du logiciel pour résoudre le problème et non l'inverse ;
- Le programmeur ne prend pas assez de recul par rapport au problème en cours de traitement. Il voit comment résoudre telle ou telle partie du problème, mais il ne sait pas appréhender le problème de manière générale et donc relier ces différentes parties entre elles. Il ne sait pas reconnaître des situations ou des sous problèmes « type » pour lesquels il dispose d'outils de résolution éprouvés. Il est souvent difficile de résoudre un problème le nez sur l'écran ;
- Le programmeur réalise un programme qui fonctionne, mais dont les qualités (lisibilité, portabilité, réutilisabilité, maintenance) et les performances (temps d'exécution et espace mémoire nécessaire) sont souvent passables voire médiocres. Ceci est la conséquence de trop de précipitation et d'un excès de confiance qui l'entraîne à corriger, bien

souvent, une erreur de programmation sans réfléchir de nouveau à la solution en reprenant l'algorithme erroné.

L'énumération de ces différents travers nous amène à définir l'algorithme comme un outil permettant la réflexion sur l'analyse d'un problème en se détachant des particularités d'un langage ou encore comme un outil permettant de définir précisément le problème à résoudre pour en arriver à une solution finale robuste afin de l'exprimer, dans en deuxième temps, dans un langage de programmation.

2. DEFINITION DE L'ALGORITHME

Vu de l'extérieur, un programme est une boîte noire dans laquelle on introduit des données d'entrée afin de produire des données de sortie. L'utilisateur ne demande qu'une chose, c'est que l'application fonctionne avec des réponse correctes obtenues dans un temps raisonnable. Le programmeur, lui doit définir le contenu de la boîte noire. Pour cela, il peut se laisser guider par son instinct et produire un programme quelle que soit la méthode utilisée. Mais ce programme est-il viable ? L'algorithme essaye de répondre à cette problématique en demandant au développeur de prendre du recul par rapport au langage de développement choisi, afin de mener une réflexion sur le futur.

L'algorithmie est une méthode de réflexion qui permet de découper un problème complexe, transformant des données d'entrée en données de sortie, en sous problèmes indépendants moins complexes pour lesquels on connaît une solution éprouvée, une solution facile à mettre en place ou encore un modèle mathématique.

Pour mieux concrétiser cette définition, prenons un exemple de problème tiré de la vie courante :

Je souhaite me rendre au concert en autobus en partant de chez moi.

Après réflexion, voyons les étapes qui vont me permettre d'atteindre mon objectif :

Début

1 Ouvrir la porte de ma maison
2 Sortir de ma maison
3 Fermer la porte de la maison à clef
4 **Tant que** je ne suis pas arrivé à l'arrêt de bus **Faire**
5 Avancer
6 **Fin tant que**
7 **Si** l'autobus est à l'arrêt de bus **alors**

```
8           Monter dans l'autobus
9       Sinon
10          Tant que l'autobus n'est pas arrivé à mon arrêt de destination Faire
11              Attendre l'autobus
12          Fin tant que
13          Monter dans l'autobus
14      Fin si
15      Tant que l'autobus n'est pas arrivé à mon arrêt de destination Faire
16          Attendre
17      Fin tant que
18      Descendre de l'autobus
19      Tant que je ne suis pas arrivée au concert Faire
20          Avancer
21      Fin tant que
22      Entrer dans le concert
Fin
```

Prenons un autre exemple :

Je souhaite faire cuire des pâtes.

Quelles sont les étapes qui vont me permettre d'atteindre mon objectif ?

```
Début
1       Sortir une casserole
2       Mettre de l'eau dans la casserole
3       Mettre la casserole sur le feu
4       Tant que l'eau ne bout pas Faire
5           Attendre
6       Fin  tant que
7       Sortir les pâtes du placard
8       Verser les pâtes dans la casserole
9       Tant que les pâtes ne sont pas cuites Faire
10          Attendre
11      Fin tant que
12      Verser les pâtes dans une passoire
13      Egoutter les pâtes
14      Verser les pâtes dans un plat
Fin
```

Ces deux exemples nous montrent que même dans nos activités de la vie courante, nous pouvons dérouler un ensemble d'actions dans un ordre précis, logique et réfléchi. Nous voyons également qu'il existe différents types d'action : des *actions conditionnées* (Si) et des *actions répétitives* (Tant que). De plus, pour décrire la succession des actions, nous avons employé *une notation simple*, lisible par tous. En fait, nous venons ni plus ni moins d'écrire un algorithme de la vie courante, qui à partir d'un point de départ nous amène à un point d'arrivée, en

déroulant différentes étapes simples. L'ensemble de ces étapes nous donne la solution à un problème plus général (aller au concert ou cuire des pâtes).

En résumé, avant de programmer une solution à un problème complexe, l'algorithmie nous demande de :

1. Découper le problème général en *sous problèmes* plus faciles à résoudre ou pour lesquels on connaît déjà une solution éprouvée.
2. Pour chaque sous problème, préciser les *données d'entrée*, préciser les *données de sortie* et spécifier le moyen de passer des données d'entrée aux données de sortie en utilisant les outils de bases de l'algorithmie (variables, instructions de lecture, d'affichage, instructions de contrôle…).
3. Vérifier le bon *fonctionnement* de chaque sous problème et valider ce sous problème (pas de boucle infinie, variables ayant les bonnes valeurs à un instant donné…). En effet, il est plus facile de construire un programme à partir de petits bout de programme fonctionnant correctement que de rechercher par la suite l'endroit du « bug » dans la globalité du programme.
4. Mettre en place l'*ordre d'exécution* des sous problèmes entre eux pour donner la solution au programme final. En bref, il faut indiquer le déroulement du programme.
5. *Vérifier la solution finale* afin de la valider. C'est à dire obtenir les bonnes données de sortie à partir de données d'entrée.
6. *Optimiser la solution* (éliminer les boucles vides, optimiser la gestion de la mémoire, ne pas laisser le programme se dérouler inutilement une fois qu'on a la réponse…).

De manière plus scientifique, Donald E.KNUTH, chercher à l'université de Stanford (USA), a défini dans l'un des trois tomes intitulés « The art of computer programming » que l'algorithmie est un ensemble de sous programmes ;

- Finis et achevés après un nombre fini d'actions ;
- Définis et précis (sans ambiguïté) ;
- Manipulant des objets définis de manière précise. Il faut toujours déclarer les objets utilisés en définissant leur nature, c'est à dire leur type ;
- Générant au moins un résultat ;
- Effectifs, c'est à dire que l'homme peut effectuer à la main l'ensemble des opérations l'une après l'autre en un temps fini.

Pour terminer, nous pouvons dire qu'un programme doit être vu en fait simplement comme une finalité, une concrétisation d'une démarche « papier ». Un algorithme est et doit être indépendant de tous langages de programmation :

- Car un langage de programmation est seulement un langage lisible qui permet de nous faire comprendre par l'ordinateur. Ce dernier ne comprend que le langage binaire (succession de 1 et 0) donc moins lisible. En fait, il existe des programmes de traduction, compilateurs ou interpréteurs, qui codent en langage machine (code binaire) ce que le programmateur a écrit ;
- Pour pouvoir être compris par tout le monde (même non informaticien) ;
- Pour éviter de construire un programme un peu au hasard. C'est à dire écrire un petit bout, le compiler, le corriger, le recompiler, lui rajouter un petit bout, le recompiler… Cela peut prendre beaucoup trop de temps et masquer des problèmes fondamentaux pour la bonne marche du programme final ;
- Ne pas construire une solution à partir des possibilités du langage mais construire une solution efficace et éprouvée ;
- Les langages sont toujours orientés vers un domaine de prédilection (FORTRAN, les calculs scientifiques, COBOL, la gestion…) et donc nous avons une vue très limitée sur la bonne solution indépendante et efficace.

3. NOTATION UTILISER EN ALGORITHMIE

Comme nous l'avons vu dans nos exemples précédents, un algorithme est écrit en *pseudo code*. Ce dernier est en fait un langage simplifié proche du langage naturel, permettant d'écrire un programme avec des mots simples. Il possède des règles d'écriture (une *syntaxe*), d'interprétation (une *sémantique*) et de présentation. L'alphabet permettant de construire les mots et les expressions est celui de la langue française (Majuscules et Minuscules autorisées), les chiffres 0-9, quelques symboles spéciaux (des signes mathématiques par exemples) et éventuellement des signes de ponctuation.

D'une manière générale, dans une première approche, un algorithme se présentera de la manière suivante :

Début

…….

……..

…….

Fin

Il faut marquer le début et la fin de l'algorithme avec les mots Début et Fin. Entre les deux, il faudra placer toutes les instructions nécessaires pour l'obtention du résultat attendu.

Reprenons notre premier exemple :

Début

5 Ouvrir la porte de ma maison
6 Sortir de ma maison
7 Fermer la porte de la maison à clef
8 **Tant que** je ne suis pas arrivé à l'arrêt de bus **Faire**
5 Avancer
6 **Fin tant que**
8 **Si** l'autobus est à l'arrêt de bus **alors**
8 Monter dans l'autobus
9 **Sinon**
10 **Tant que** l'autobus n'est pas arrivé à mon arrêt de destination **Faire**
12 Attendre l'autobus
12 **Fin tant que**
13 Monter dans l'autobus
14 **Fin si**
15 **Tant que** l'autobus n'est pas arrivé à mon arrêt de destination **Faire**
16 Attendre
17 **Fin tant que**
18 Descendre de l'autobus
19 **Tant que** je ne suis pas arrivée au concert **Faire**
20 Avancer
21 **Fin tant que**
22 Entrer dans le concert

Fin

Les numéros qui se trouvent devant chaque instruction ne sont là que pour faciliter le repérage des lignes dans le cadre des explications qui vont suivre mais ils ne son absolument pas obligatoires.

Avant de détailler les outils de l'algorithmie, voici quelques remarques sue l'exemple précédent :

- Lignes 1-3 : Il s'agit d'un ensemble d'*instructions simples* qui ne s'exécutent qu'une seule fois au cours du déroulement de l'algorithme.

- Lignes 4-6 : Il s'agit d'une *boucle* ou d'une *répétitive*. En effet tant que la condition est vraie, à savoir « je ne suis pas arrivée à l'arrêt de bus », on exécute en boucle les instructions situées entre le « Tant que » et le « Fin tant que ». Lorsque la condition devient fausse, à savoir « je suis arrivé à

l'arrêt de bus », l'algorithme continue son exécution à partir de la ligne qui suit le « Fin tant que », donc la ligne n°7.

- Lignes 7-14 : Il s'agit d'une condition. Si le résultat du test est vrai, on exécute les instructions du bloc « Alors (ligne 8) ». Si le résultat du test est faux, on exécute les instructions du bloc « Sinon (lignes 10-13) ». Le test ne s'exécute qu'une seule fois au cours du déroulement de l'algorithme. En détaillant un peu plus, si le test de la condition donne vrai c'est que le bus est là et donc je peux monter dedans. Si le test de la condition donne faux, il faut donc attendre que le bus arrive puis lorsqu'il est là, je peux monter dedans.

- Lignes 10-13 : Ce bloc se compose d'une boucle (lignes 10-12) et d'une instruction simple (ligne 13). L'algorithme boucle tant que la condition est vraie, à savoir « l'autobus est à l'arrêt de bus », la boucle se termine et on exécute l'instruction qui suit le « Fin tant que », donc on monte dans le bus.

- Lignes 15-17 : Il s'agit de nouveau d'une boucle ou d'une répétitive. Lorsque le test de la condition devient faux, à savoir « l'autobus est arrivé à mon arrêt de destination », l'algorithme se poursuit à la ligne qui suit le « Fin tant que », à savoir la ligne n°18.

- Ligne 18 : Il s'agit d'une instruction simple qui ne s'exécute qu'une seule fois au cours du déroulement de l'algorithme.

- Lignes 19-21 : Il s'agit d'une boucle ou d'une répétitive. Lorsque le test de la condition devient faux, à savoir « je suis arrivé au concert », l'algorithme se poursuit à la ligne qui suit le « Fin tant que », à savoir la ligne n°22.

- Ligne 22 : Il s'agit d'une instruction simple qui ne s'exécute qu'une seule fois au cours du déroulement de l'algorithme.

Toutes ces différentes familles d'instructions seront détaillées dans le cours suivant « Outils de base », mais nous pouvons déjà observer qu'elles respectent une certaine syntaxe et surtout une certaine mise en forme que l'on appelle l'*indentation*. Cette indentation décale les lignes, de manières à rendre plus lisible un programme. C'est essentiel.

4. LES DIFFERENTES PHASES DE CREATION D'UN PROGRAMME

l'écriture d'une application informatique comporte plusieurs phases progressives :

4.1 Phase de spécification

cette phase permet à chaque partenaire de se mettre d'accord sur la finalité du programme en cours de création. Au cours de réunions avec les futurs utilisateurs, il faut définir de manière exacte :

- Les spécifications d'entrée : quelles sont les informations données au départ et avec quelles contraintes (origine des données, type, valeur…) ;
- Les spécifications de sortie : l'utilisateur souhaite obtenir quels types de réponses et sous quelles formes ;
- Les spécifications de traitement : il est nécessaire de déterminer les traitements à effectuer, les règles de gestion à respecter, les cas particuliers et enfin les réactions en cas d'erreur.

Cette phases aboutit à la rédaction d'un cahier des charges qui doit être signé par toutes les parties.

4.2 Phase de conception

C'est dabs cette phase qu'intervient l'algorithmie. En effet, lors de cette étape une analyse structurée du problème afin de rechercher une solution globale. Pour cela, il faut essayer de découper le problème en problèmes plus petits que l'on peut encore, le cas échéant, redécouper. Pour chacun de ces sous problèmes, on va essayer de le ramener à un problème connu. Attention, le fait de repérer la solution à un sous problème n'implique pas de disposer d'une méthode pour réaliser la solution. En effet une équation mathématique ne fournit pas forcément un moyen évident de résolution informatique.

Le travail de résolution consiste donc à déterminer les enchaînements d'opérations produisant les données de sortie à partir des données d'entrée. De la solution n'est pas facile à mettre en place, on essaye alors de simplifier le modèle mathématique en le remplaçant par un modèle approchant pour lequel on dispose d'une méthode de résolution.

Il s'agit d'une analyse descendante (du plus général au plus particulier au « top down » en anglais). Dans cette phase nous utilisons le pseudo code. Il sera également nécessaire de définir les liens entre chaque sous problème.

4.3 Phase de codage

Une fois que les différents problèmes ont leur solution algorithmique, il faut écrire le programme dans un langage de programmation. Cela nous amène parfois à rencontrer des problèmes liés aux contraintes du langage utilisé et donc à revoir certaines parties de l'algorithme. Dans ce derniers cas, il est nécessaire de refaire l'analyse papier pour garder la cohérence de notre solution.

4.4 Phase de correction

Dans cette étape, il faut définir des *jeux d'essais* prenant en compte tous les cas d'erreurs possibles. Ces tests de validation permettent la mise au point du programme. C'est la phase la plus longue. On distingue les erreurs de syntaxe détectées lors de la phase de compilation, qui se corrigent facilement, des erreurs d'exécution entraînant un comportement anormal lié dans certains cas à un problème de conception. Dans cette dernière hypothèse, il est nécessaire de repartir de l'algorithme papier pour le modifier éventuellement.

4.5 Phase de maintenance

Les programmes vivent et vieillissent. Cela nécessite parfois une mise à jour du programme afin de le tenir à jour par rapport à des nouveaux besoins. A ce stade, on voit encore plus la nécessité d'un travail bien préparé afin de minimiser les modifications. Le temps, c'est de l'argent !