Contenu du cours

- Le problème de la programmation
- Description d'un langage algorithmique
- Programmation structurée : conception d'un algorithme arborescent
- Analyse modulaire : conception d'architectures fonctionnelles
- Organisation des données d'un algorithme : types abstraits et constructeurs de types
- Analyse modulaire : conception d'architectures modulaires et orientées objets
- Codage et mise au point des programmes

Notion d'algorithme et de processeur

- **Un processeur** est une entité capable de comprendre un algorithme
- L'exécution effective, au cours du temps, de l'algorithme par le processeur s'appelle **un processus**
- On appelle **objets** manipulés par un processeur les (classes d') objets sur lesquels le processeur sait réaliser directement une action
- On appelle **actions élémentaires** ou **primitives du processeur** les actions que le processeur sait réaliser directement dès lors que lui est énoncée une telle action et lui sont désignés les objets sur lesquels elle porte

Algorithme	Processeur	Objet manipulé	Processus
Méthode d'addition	Enfant	Chiffres	Réalisation d'une addition
Partition	Musicien	Instrument	Jeu de la partition
Recette	Cuisinier	Casserole	Confection du plat

Définition d'un algorithme

- Etant donné une action abstraite à réaliser et un processeur défini par l'ensemble des (classes d')objets qu'il sait manipuler et la liste de ses actions primitives, on appelle **algorithme** l'énoncé de l'ensemble (structuré) d'actions primitives permettant de réaliser cette action, chaque énoncé d'action primitive comportant la désignation des objets sur lesquels elle doit porter
- Si le processeur correspond à un langage de programmation, l'algorithme est également appelé **un programme**

Niveau d'abstraction	Processeur	Etat de la conception
élevé	cuisinier averti	recette 1
moyen	cuisinier ordinaire	recette 2
final	novice	recette 3

Ecriture d'un algorithme et codage d'un programme

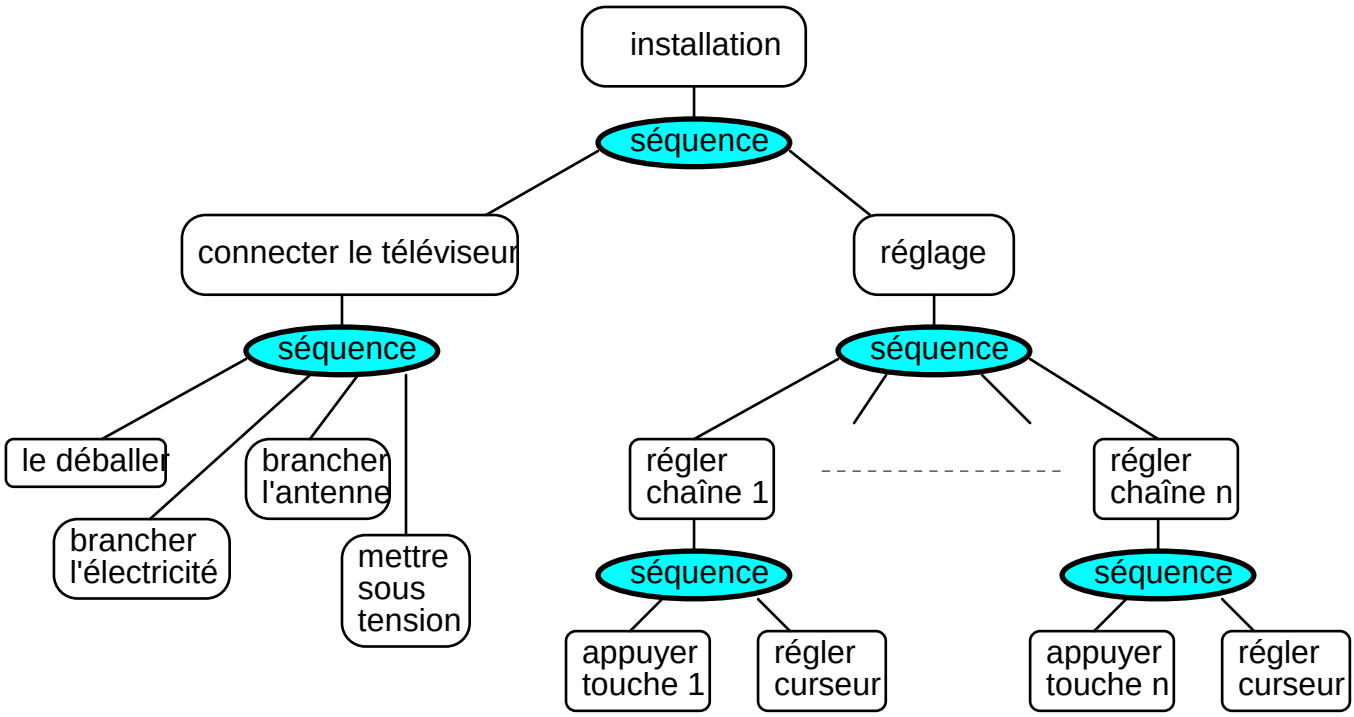
- On appelle langage algorithmique un langage permettant d'écrire sous forme d'algorithmes les différents états intermédiaires par lesquels passe un programme au cours de sa conception, (presque) indépendamment du langage visé
- La conception de ces algorithmes successifs s'appelle l'analyse
- La réécriture dans un langage de programmation particulier d'un algorithme exprimé en langage algorithmique s'appelle le codage
  - ◊ Intérêt d'un langage algorithmique
  - ◊ Il permet de raisonner par niveaux d'abstraction décroissants
  - ◊ Il assouplit les contraintes syntaxiques
  - ◊ Il unifie la méthode de conception des programmes, quel que soit le langage visé
- Il permet de fractionner la difficulté de conception en séparant l'analyse et le codage

Conception d'un algorithme : méthode d'analyse descendante

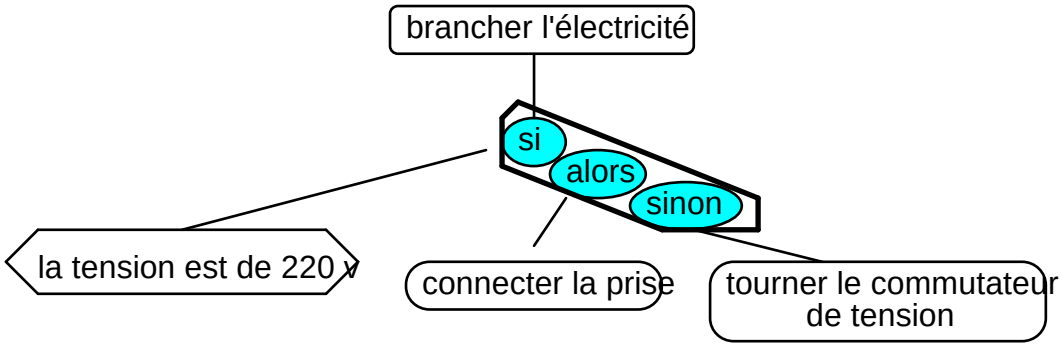
- **Rôle d'une méthode de programmation**
  - ◊ Simplifier
  - ◊ Rendre rigoureuse la démarche de conception d'un programme
- **L'analyse descendante**
  - ◊ On définit en premier lieu, avec précision, l'action abstraite qu'il s'agit de réaliser : propriétés qu'elle doit vérifier, objets sur lesquels elle porte ; c'est ce que l'on appelle la spécification externe ou spécification de l'action
  - ◊ On décompose chaque action abstraite en actions abstraites de niveau plus simple. Cette décomposition qui permet de définir un nouvel algorithme porte le nom de raffinage
  - ◊ On recommence les étapes 1 et 2, jusqu'à l'obtention d'actions primitives, obtenant ainsi l'algorithme final

Représentation arborescente

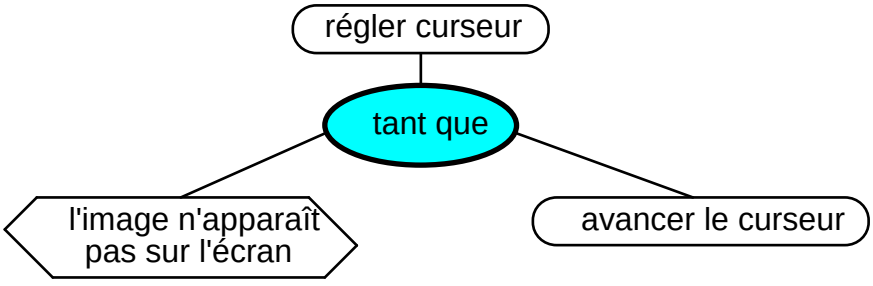
Un énoncé séquentiel



Un énoncé conditionnel



Un énoncé répétitif



Terminologie

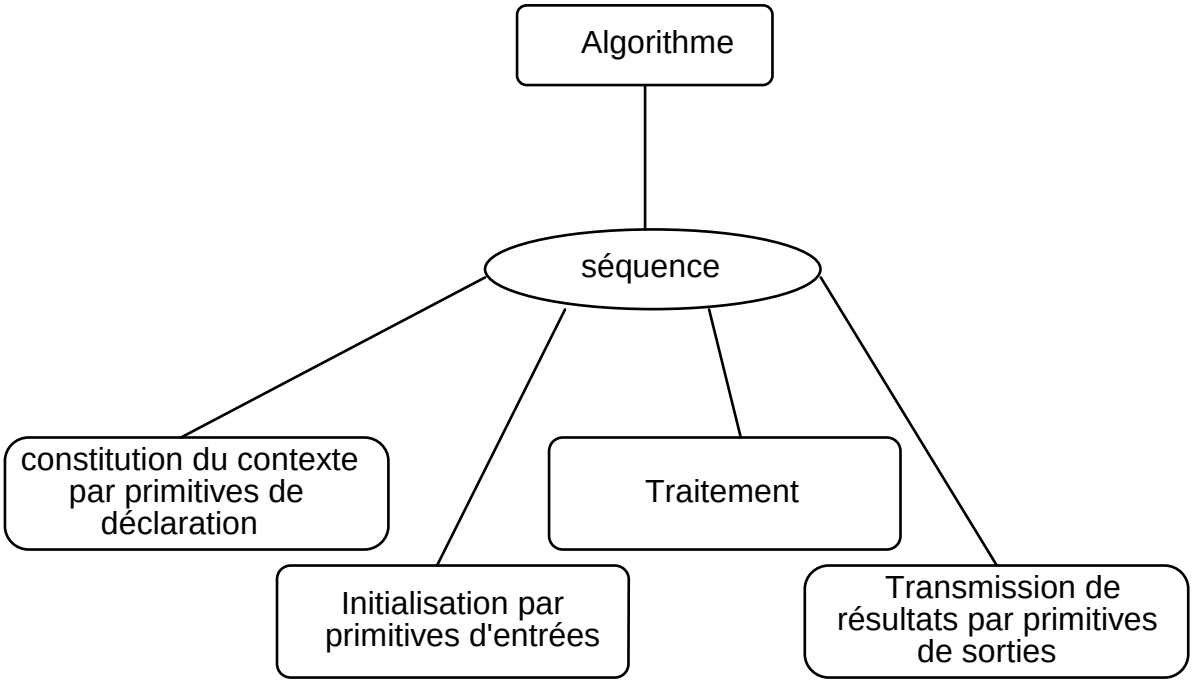
- Les trois schémas de décomposition : **séquence**, **alternative** et **répétition**, sont suffisants pour énoncer n'importe quel algorithme. Il en résulte que tout processeur devra être capable de les comprendre, et que tout langage de programmation sera capable de les coder
- Lorsque les actions abstraites qui interviennent dans un algorithme ne sont pas spécifiées, il est beaucoup de questions auxquelles il est impossible de répondre. Le simple énoncé du nom d'une action à réaliser est loin d'être suffisant pour définir avec précision la situation dans laquelle on se trouvera lorsque l'on voudra réaliser l'action suivante
  - ◊ **Spécification** : description d'une action en termes d'état initial et d'état final
  - ◊ **Action spécifiée** : abstraction sur laquelle on peut raisonner

Objets mis en oeuvre dans un algorithme

- On appelle **contexte d'exécution** d'un algorithme l'ensemble des objets accessibles au processeur pendant le déroulement de l'algorithme. Chacun de ces objets possède trois caractéristiques :
  - ◊ **un nom** qui permet de le désigner,
  - ◊ **un type** qui fixe l'ensemble des valeurs qu'il peut prendre et les actions qui peuvent porter sur lui
  - ◊ **une valeur** sur laquelle va effectivement porter l'action du processeur
- On appelle primitives de déclaration les primitives permettant de déclarer à un processeur les objets auxquels il peut accéder, en précisant pour chaque objet :
  - ◊ le nom par lequel il sera désormais désigné
  - ◊ le type auquel il appartient
- Ces primitives seront toujours les premières d'un algorithme

Echange avec l'extérieur

- On appelle **primitives d'entrée-sortie**, les primitives particulières qui permettent à un processeur :
  - ◊ de demander une valeur à son environnement et de la ranger dans un de ses objets
  - ◊ de communiquer à son environnement la valeur d'un de ses objets
- On appelle **environnement d'un algorithme** toute entité susceptible de fournir à un processeur, sur sa demande, des valeurs, ou de recevoir des valeurs



Isolement des actions d'entrée-sortie

- **Les primitives d'entrée-sortie ne dépendent pas du processeur**
  - ◊ Elles peuvent provoquer une erreur même si l'algorithme est correct
  - ◊ Elles doivent être modifiées chaque fois que l'environnement dans lequel s'exécute l'algorithme est modifié
- **Il faut , pour éviter un blocage**
  - ◊ préciser soigneusement, au niveau de l'algorithme, les ensembles auxquels doivent appartenir les valeurs fournies par l'environnement
  - ◊ protéger l'exécution du programme des erreurs provenant de l'environnement en ajoutant à toute action d'entrée-sortie un fragment de programme chargé de vérifier que les valeurs fournies appartiennent bien aux ensembles autorisés

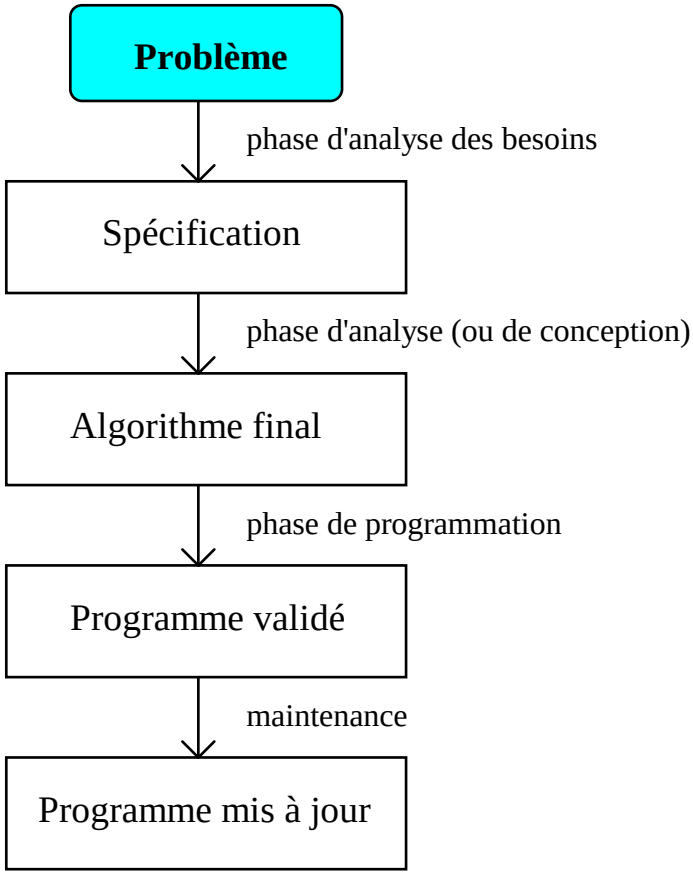
Situation statique d'un algorithme

- On appelle **situation statique** d'un algorithme l'état dans lequel se trouve son contexte à un moment de son exécution
- On appelle **assertion** une affirmation décrivant la situation statique d'un algorithme en un point donné sous forme d'une relation logique portant sur les objets de son contexte
- **Exemples d'assertions :**
  - le téléviseur est sous tension
  - le téléviseur visualise l'image de la chaîne 2
  - unité_1 = 3 ; dizaine_1 = 2 ; unité_2 = 7 ; dizaine_2 = 5 ;
- **Les avantages de l'assertion**
  - Elle permet de décrire les situations par lesquelles passe un programme
  - Elle facilite la conception, la compréhension, la mise au point d'un programme

Spécification d'une action

- **Intérêt :** rendre l'action indépendante du reste de l'algorithme
  - Exemple :
    - [le téléviseur est dans son carton, près de la porte]
    - installer le téléviseur
    - [le téléviseur est en place, toutes les chaînes sont réglées, il fonctionne sur la chaîne 3]
  - Action :
    - nécessite : (précondition)
    - entraîne : (postcondition)

Le cycle de vie d'un programme



La méthode scientifique en programmation

- **Principe 1**
  On doit pouvoir raisonner de façon rigoureuse sur chacune des représentations intermédiaires utilisées lors de la conception d'un programme afin d'identifier immédiatement une erreur éventuelle
- **Principe 2**
  Un programme n'est pas seulement une méthode de commande d'un processeur, il est surtout un outil de communication entre des hommes : programme et documentation sont une seule et même chose
- **Lire un programme**
  Lire la succession de représentations intermédiaires utilisées lors de sa conception
- **Modifier un programme**
  Modifier son analyse, modifier toutes les représentations intermédiaires concernées

Définition d'un langage algorithmique

- **Identificateurs**
  - Ne jamais utiliser un mot réservé du langage
  - panier, sup2
  - Préciser la nature de la variable
    - chaîne, entier, booléen
- **Chaine constante**
  - "donner le prix de l'article"
- **Entree**
  - lire prix1, prix2
- **Sortie**
  - ecrire "le prix est", prix
- **Affectation**
  - identificateur ← identificateur
  - identificateur ← valeur
    - sup ← 0
- **Séparateur d'instructions**
  - Pas de séparateur d'instruction
  - spécifié (mais on peut utiliser le ; pour séparer les instructions si on le désire)
- **Opérations**
  - Arithmétiques : + , - , * , / , % (modulo)
  - comparaisons : < , > , <= , >= , = = (=) , != (≠)
  - logiques : && (et) , || (ou) , ! (not)
  - tableau : identificateur [indice]
- **Structure alternative**
  - Si condition (ex: Si valeur = 5)
  - Alors
    - Action 1
  - Sinon
    - Action 2
  - fin si
- **Structure d'aiguillage**
  - au cas où :
    - variable1 = valeur :
      - instruction 1 à n
    - variable n = valeur :
      - instruction 1 à m
- **Structure répétitive (1)**
  - tant que condition
  - faire
    - Action
  - fin tant que
- **Structure répétitive (2)**
  - répéter
    - Action
  - tant que condition
  - fin répéter
- **Structure répétitive (3)**
  - pour i variant de val1 à val2
    - instruction
  - fin pour
- **Commentaires**

/* ceci est un commentaire */

nota : Les opérations entre ( ) peuvent aussi s'écrire.
ex. soit : *a et b* soit : *a && b*

Principes méthodologiques de l'analyse modulaire

- **Principe d'abstraction**
  Chaque unité modulaire doit correspondre à une abstraction préexistante et doit pouvoir être définie de façon abstraite, indépendamment de sa réalisation
- **Principe d'encapsulation**
  La description logique d'une unité modulaire doit permettre de cacher systématiquement toute sa complexité interne tout en fournissant les informations suffisantes pour la comprendre et l'utiliser
- **Principe de faible couplage**
  A chaque niveau d'analyse, les rapports entre unités modulaires qu'il est nécessaire de considérer doivent être simples

Principe d'encapsulation

- **Toute unité d'un programme modulaire fait l'objet de deux représentations**
  - ◊ **une représentation externe**, publique, regroupant les seules informations nécessaires pour la comprendre et l'utiliser. Cette représentation publique, appelée interface logique, décrit, de façon externe, à la fois :
    - sa sémantique,
    - ses rapports avec l'extérieur,
  - ◊ **une représentation interne**, cachée, qui constitue la réalisation effective de l'unité est appelée corps de l'unité modulaire

Principe de faible couplage

- **Principes de faible couplage**
  - A chaque étape de l'analyse, le nombre d'unités modulaires à considérer doit rester "petit"
  - Les rapports entre ces unités sont simples : il y a peu d'informations échangées, et ces échanges ont lieu entre peu d'unités
- **L'analyse modulaire**, procédant par identification d'abstractions, précède normalement l'analyse structurée qui proçède, au niveau de chaque unité modulaire, par décomposition d'actions
- **Les sous-programmes** constituent la classe la plus simple d'unités modulaires

Preuve d'un algorithme modulaire

- **La preuve d'un algorithme modulaire** est exécutée unité par unité. Elle consiste à prouver séparément pour chaque unité que :
  - si chaque sous-programme appelé respecte ses spécifications,
  - alors l'unité en cours, elle-même, respecte les siennes
- **Le domaine de valeur des paramètres d'entrée** pour lequel le comportement d'un sous-programme est spécifié doit être tel que l'appelant puisse effectivement vérifier que les valeurs qu'il fournit appartiennent à ce domaine
- Lorsque l'on souhaite une **généricité** plus large, la méthode consiste à écrire un modèle d'algorithme. Puis, à générer autant d'algorithmes exécutables qu'il est nécessaire pour traiter les différents types

Le logiciel, état des lieux

- **Les temps de conception et de développement sont longs**
  - évolution du besoin, phase de conception souvent peu tangible
- **La communication et l'implication des utilisateurs pendant le développement passent pour rester de faible niveau**
  - le langage de l'informaticien et celui de l'utilisateur final
- **Les tests du logiciel restent un des grands points noirs de l'informatique**
  - la testabilité doit faire partie de la conception
- **La maintenance prend une part de plus en plus importante des ressources informatiques**
  - structure complexe du logiciel et dépendances inextricables
- **Les besoins s'élargissent et les exigences s'accroissent**
  - ergonomie, performance
- **L'évolution des techniques introduit davantage de complexité qu'elle n'en supprime**
  - multimédia, interfaces graphiques

Pourquoi la programmation est intrinsèquement complexe ?

- **Désadaptation d'impédance entre les utilisateurs d'un système et ses développeurs**
- **Les spécifications d'un logiciel changent souvent en cours de développement**
- **La tâche de l'équipe de développement est de donner l'illusion de la simplicité**
- **Il faut que les logiciels offrent une flexibilité absolue**
- **Problèmes de la caractérisation des systèmes discrets**
  - des centaines ou même des milliers de variables
  - les valeurs et adresses des variables évoluent

Les conséquences d'une complexité sans limite

- Plus un système est **complexe**, plus il est susceptible **d'effondrement**
- Gaspillage des ressources humaines
- Comment résoudre cette crise du logiciel?
- Comment sont organisés les systèmes complexes dans d'autres disciplines?
- Le programmeur amateur réalise des logiciels à durée de vie limitée
- Exemples d'application complexe :
  - systèmes gérant des entités du monde réel
    - » contrôle de trafic aérien
  - systèmes simulant l'intelligence humaine
    - » réseaux de neurones
- La complexité de ces systèmes dépasse les capacités intellectuelles de l'homme
- Il faut trouver des méthodes rigoureuses pour maîtriser la complexité

Exemple de système complexe: la structure d'un ordinateur personnel

- **Les éléments d'un ordinateur personnel sont :**
  - une unité centrale, un écran, un clavier, un disque dur
- **Une unité centrale englobe:**
  - une mémoire centrale
  - une unité arithmétique et logique
  - un bus
- **Une unité arithmétique et logique englobe:**
  - des registres et une logique de contrôle
- **Nature hiérarchique d'un système complexe**
- **L'ordinateur ne peut fonctionner qu'à partir du moment où ses parties collaborent**
- **Chaque partie peut être indépendante**

Les 5 attributs d'un système Complexe

- **La complexité** prend souvent la forme d'une hiérarchie dans laquelle un système complexe est composé de sous-systèmes reliés entre eux et ayant à leur tour leurs propres sous-systèmes,et ainsi de suite jusqu'à ce qu'on atteigne le niveau le plus bas des composants élémentaires
- **Le choix** des composants primaires d'un système est relativement arbitraire et dépend largement de l'observateur du système
- **Les liaisons intra-composants** sont généralement plus fortes que les liaisons **inter-composants**. Ceci a pour effet de séparer les dynamiques haute fréquence des composants - celles qui concernent la structure interne des composants - des dynamiques basse fréquence - qui concernent l'intéraction entre composants
- **Les systèmes hiérarchiques** sont habituellement composés d'un petit nombre de genres de sous-systèmes qui forment des combinaisons et des arrangements variés
- **Un système complexe** qui fonctionne a toujours évolué à partir d'un **système simple** qui a fonctionné...Un système complexe conçu ex nihilo ne fonctionne jamais et ne peut être rapiécé pour qu'il fonctionne. Il faut tout recommencer, à partir d'un système simple qui fonctionne.

Les méthodes de conception

- **Principe de base : la décomposition**
  - La taille et la complexité des logiciels imposent la décomposition du problème en sous-problèmes, jusqu'à atteindre des unités plus facilement maîtrisables
- **Approche structurée**
  - Guidée par les traitements - module principal, décomposition progressive..., on obtient des modules fonctionnels (FORTRAN)
- **Approche par les données**
  - Analyse en structure de données (Warnier, Cobol)
- **Approche par les connaissances**
  - Réservées au développement de systèmes experts mais adaptées dans tous les cas où le domaine se caractérise par une forte expertise
- **Approche objet**
  - Conception d'unités autonomes qui encapsulent à la fois les données et les traitements qui s'y rapportent
- **Exemple de l'approche structurée :**
  - Pour construire une remise :
    - a) mettre en place les fondations
    - b) bâtir les murs
    - c) mettre en place le plancher
    - d) placer le toit dessus

Evolution de la séparation données-traitements

- **1re époque**
  - priorité aux traitements, justifiée par les faibles performances des ordinateurs
- **2e époque**
  - priorité aux données, les données étant plus stables que les traitements
- **3e époque**
  - approche mixte données-traitements, mixte mais séparée : les modèles sont séparés, les activités distinctes (Merise)
- **4e époque**
  - encapsulation, les données et les traitements sont appréhendés simultanément à travers des unités insécables
- **5e époque**
  - fusion de l'approche objet avec l'approche par les connaissances
- **Conclusion :**
  - L'approche objet doit-elle tirer un trait sur les méthodes antérieures ?

Limites des approches traditionnelles

- **Taille et complexité croissantes des logiciels**
- **Manque de localisation**
  - Eparpillement à travers les programmes des différents aspects concernant une même entité (cet éparpillement est lié d'abord à la séparation données-traitements, et s'aggrave par la répartition des traitements)
- **Faible réutilisabilité**
- **Difficulté d'évolution**
- **Primat de l'action (= traitement)**
  - Ce qui est vrai pour les méthodes par décomposition fonctionnelle
- **Difficulté dans le traitement du parallélisme**
- **Logique étrangère à la réalisation des interfaces utilisateur**
- **Conclusion :**
  - Le bon sens réclame la meilleure adéquation possible entre le modèle informatique et la réalité

Quelle décomposition choisir ?

- **Décomposition algorithmique**
  - Dogme de la programmation structurée
- **Décomposition orientée objets**
  - Série d'agents autonomes qui collaborent pour réaliser un certain comportement de plus haut niveau
- **Décomposition algorithmique et orientée objets:**
  - La vue algorithmique souligne l'ordre des événements
  - La vue orientée objets met l'emphase sur les agents qui causent une action ou sont les sujets sur lesquels ces opérations agissent
  - La vue orientée objets organise mieux la complexité inhérente aux logiciels
  - La vue orientée objets permet la ré-utilisation de mécanismes communs

Approche de la définition des objets

- **L'espace de problème**
  C'est l'environnement dans lequel se déroule l'informatisation, environnement à la fois organisationnel et technique, mais surtout sémantique ou culturel
- **L'espace de solution**
  C'est le discours dans lequel s'élabore la solution. Il est caractérisé par une modélisation fondée sur des structures de représentation et contrainte par des catégories a priori
- **Modéliser, c'est mentir**
  Comment limiter, voire supprimer la distorsion entre l'espace de problème et l'espace de solution ? Comment maîtriser le processus mental de la modélisation ?
- **La réponse :**
  L'utilisateur perçoit dans sa pratique des "entités" stables, douées d'autonomie, et qui se manifestent par des informations accessibles et des comportements, soit : des objets

Exercices

1) Ecrire l'algorithme de l'addition de 2 nombres à 2 chiffres en utilisant une notation graphique.

2) Ecrire l'algorithme de la multiplication de 2 nombres à 2 chiffres en utilisant une notation graphique.

3) On suppose que nous possédons, vous et moi, autant d'argent l'un que l'autre. Combien dois-je vous donner pour que vous ayez 10F de plus que moi ?

4) Vous payez 20 F une bouteille de beaujolais. Le vin coûte 19 f de plus que la bouteille. Combien vaut la bouteille ?

5) Il faut 56 biscuits pour nourrir 10 animaux. Ces animaux sont des chats et des chiens. Un chien mange 6 biscuits et un chat n'en mange que 5. Parmi ces animaux, combien sont des chats et combien sont des chiens ?

6) Une machine est alimentée à partir de 2 entrées E1 et E2. On demande d'envoyer les boules blanches dans le récipient S1, les boules rouges de E1 dans le récipient S2, les boules de E2 dans le récipient S3 et le reliquat dans le récipient S4. On traitera d'abord toutes les boules de E1, puis, lorsque celui-ci sera vide, celles de E2. On devra programmer la machine pour qu'elle s'arrête lorsqu'elle tentera d'alimenter une boule à partir de E2 vide
Jeux d'essai : E1 : OBRRN E2 : VMRBB

7) Une machine est alimentée à partir de 2 entrées E1 et E2. Le premier réservoir d'alimentation contient des boules rouges, une boule verte, des boules blanches. Le deuxième contient des boules de différentes couleurs.
On demande de mettre :

- les boules blanches de E1 et les boules blanches de E2, s'il y en a, dans le récipient d'évacuation S1
- les boules rouges de E1 et les boules vertes de E2, s'il y en a, dans le récipient d'évacuation S2
- la boule verte de E1 et le reliquat de E2 dans le récipient d'évacuation S3

On commencera par alimenter uniquement E1 jusqu'à ce que la boule verte soit arrivée à destination, ensuite on alimentera exclusivement E2. On devra programmer la machine pour qu'elle s'arrête lorsqu'elle tentera d'alimenter une boule à partir d'un réservoir vide.
Jeux d'essai : 1) E1 : RRBVBR E2 : BBVRV
2) E1 : RRBV E2 : BVR

8) Ecrire la permutation de deux nombres

9) Permutation circulaire de trois nombres

10) Composer un algorithme qui lit trois noms frappés successivement au clavier d'un poste de travail puis affiche à l'écran ces trois noms, dans l'ordre alphabétique

11) Trois valeurs numériques ont été lues dans trois variables numériques x, y, et Z. Composez une séquence qui calcule , dans une variable numérique C, le nombre de valeurs distinctes (3, 2, ou 1)

12) On a relevé sur une feuille une liste comportant 40 noms. Composez un algorithme qui effectue la saisie de ces noms, dans l'ordre dans lequel ils apparaissent sur cette liste, puis affiche ces noms à l'écran, dans l'ordre inverse.

13) On reprend la liste de noms. Composer un algorithme qui saisit et affiche les noms, dans l'ordre dans lequel ils apparaissent sur la liste, à raison de deux noms par ligne d'écran. Si le nombre de noms est impair, la dernière ligne ne contiendra qu'un nom.

14) Toujours à partir de la liste de noms, on veut :
- Faire la saisie des noms dans l'ordre dans lequel ils se présentent sur la liste,
- Afficher ces noms à l'écran sur deux colonnes, de telle sorte qu'ils apparaissent dans l'ordre de la saisie quand on lit d'abord la première colonne, puis la deuxième.
- Composez l'algorithme nécessaire. Pour affichez deux colonnes, il suffit de grouper deux noms par lignes.

15) On reprend la liste de noms (40 au maximum). Composez un algorithme qui saisit ces noms puis les affiche dans l'ordre alphabétique. Pour obtenir l'ordre alphabétique, on placera les noms dans un vecteur en rangeant chaque nom saisi à sa place par rapport aux noms déjà rangés. Par exemple, si *'chabert'*, *'magnard'* et *'perrin'* sont déjà placés dans les éléments de rang 1, 2 et 3 et si dupond est le quatrième nom saisi, on décalera *'magnard'* et *'perrin'* respectivement dans les éléments de rang 3 et 4 et l'on placera *'dupond'* dans l'élément de rang 2.

16) Avec le processeur que l'on utilise, on suppose que l'on dispose d'une fonction permettant de tirer un entier au hasard dans un intervalle [d, f] sous la forme d'une fonction nommée hasard(d, f) (d et f sont deux entiers tels que d < f). Par exemple, si X est une variable numérique, chaque exécution de X = hasard(1, 10) affectera à X un entier tiré au hasard parmi les nombres de 1 à 10.
Composez un algorithme qui étudie la qualité de la répartition au hasard ainsi obtenue pour la simulation de jets de dés (tirage de 1 à 6). L'algorithme doit effectuer un grand nombre de tirages et afficher le nombre d'apparitions de chaque entier possible : si la répartition est bonne, ces nombres doivent être sensiblement égaux.

17) On veut programmer un jeu de dés très simple, opposant un joueur à l'ordinateur. Le joueur et l'ordinateur jettent alternativement le dé (le jet de l'ordinateur est simulé par un tirage au hasard et les points marqués à chaque jet sont cumulés pour chaque participant au jeu jusqu'à ce que l'un des cumuls dépasse 30. Le joueur ayant ce cumul est alors déclaré vainqueur.
Composez l'algorithme correspondant en étudiant bien les procédures de dialogue entre le joueur et l'ordinateur.

18) Ecrire un algorithme permettant de compter le nombre de mots dans une chaîne fournie. Les mots peuvent être séparés par un ou plusieurs espaces et on peut rencontrer des caractères de ponctuation tels que virgules, points, points d'exclamation, etc ... qui doivent être traités au même titre que les espaces.

19) Composez un algorithme permettant de chercher, dans une chaîne de caractères, le n-ième mot pour un n donné. Si la chaîne ne comporte pas n mots, on rend, comme résultat, la chaîne vide.

20) On a besoin, pour analyser des phrases, d'une fonction SUITE, permettant la vérification de l'existence dans une chaîne de deux sous-chaînes dans un ordre défini, mais pas forcément consécutives. Par exemple, si l'on applique SUITE('un', 'peu') aux chaînes suivantes, on obtiendra les réponses indiquées :

- 'donne moi un peu de temps' vrai
- 'ça fait un peu mal' vrai
- 'c'est peu, un franc' faux

Rappel : on dispose d'une fonction rang (contenant, contenu) qui fournit la position du premier caractère de la première apparition de contenu dans contenant ou 0 si contenu est absent de contenant.

21) Tous les langages de programmation permettent l'introduction de nombres frappés au clavier de l'ordinateur. Ces nombres sont, dans un premier temps, stockés sous la forme d'une chaîne de caractères qui doit être ensuite analysée et convertie en une valeur numérique. Supposez que vous devez travailler avec un ordinateur sur lequel cette possibilité n'existe pas. Il faut donc la réaliser vous-même.
Ecrivez une séquence permettant la recherche du premier nombre entier présent dans une chaîne. Le nombre peut être précédé par des caractères non numériques qu'il faut ignorer. Dès qu'on rencontre un chiffre ou un signe (+ ou -), il faut déchiffrer la suite jusqu'à la rencontre d'un nouveau caractère non numérique ou de la fin de la chaîne.

22) Un éditeur a besoin de vérifier les manuscrits qu'on lui soumet, afin d'éviter les fautes d'orthographe et la répétition abusive de certains mots. Danns ce but, on vous demande d'écrire un algorithme qui lit un texte, ligne par ligne, au clavier. Au fur et à mesure de l'introduction du texte, le programme comptabilise le nombre de fois que chaque mot a été utilisé. Lorsque tout le texte a été lu, on affichera la liste, en ordre alphabétique, de tous les mots différents qui ont été trouvés, ainsi que le nombre de fois qu'on aura trouvé chaque mot. On suppose que la dernière ligne du texte est composée uniquement d'un point, suivi d'un retour chariot.

23) Il existe un jeu de 52 cartes appelé "black-jack". Les règles de ce jeu sont très simples, et la stratégie également. Il y a un banquier et un joueur. Le banquier distribue quatre cartes alternativement au joueur et à lui-même (deux à chaque participant). La valeur des cartes est

- 10 pour les figures
- la valeur nominale pour les autres cartes, excepté l'as

1 ou 11 au choix pour l'as, à la convenance du joueur, qui peut donc à tout moment faire ses comptes avec l'une ou l'autre de ces valeurs.
Le but est d'obtenir le plus grand score sans dépasser 21. Si, avec les deux cartes distribuées, le joueur estime n'avoir pas assez de points, il peut demander des cartes supplémentaires. Ces cartes sont déposées à découvert sur la table, alors que les deux premières restent cachées. Lorsque le joueur estime que son jeu est satisfaisant, il le signale et c'est au tour du banquier de reprendre des cartes s'il le désire. Au cas où le joueur dépasse les 21 points, il doit le signaler et ne plus prendre des cartes. Dans ce cas, le banquier n'a évidemment, pas intérêt à continuer, car il a gagné de toute façon. La partie s'arrête lorsque les deux adversaires estiment avoir un score convenable. On abat alors les cartes et on fait les comptes, afin de savoir qui a gagné. A la fin de chaque partie, on peut décider de s'arrêter ou de poursuivre le jeu. En cas de poursuite, on joue avec les cartes qui restent dans le jeu. Lorsque le jeu est épuisé, ce qui peut arriver en cours d'une partie, on bat un nouveau jeu de cartes et on continue la partie. On totalise le nombre de parties gagnées et perdues et le gagnant final est celui qui a remporté le plus grand nombre de parties.
Composez un programme permettant de jouer au black-jack.

24) Le jeu de mastermind est un jeu dans lequel il faut découvrir une suite ordonnée de m couleurs prise parmi n (une couleur pouvant se trouver plusieurs fois dans la suite).
Le meneur de jeu propose une suite ordonnée de m couleurs qui reste cachée. Le joueur va essayer de la découvrir en proposant successivement des suites ordonnées de m couleurs. Lorsque le joueur propose une suite de m couleurs, le meneur de jeu compare la proposition du joueur avec la suite cachée et indique le nombre de couleurs exactes et bien placées ainsi que le nombre de couleurs exactes mais mal placées. Ecrire un programme qui, étant donné la suite de couleurs cachée, réalise le travail du meneur de jeu pour chaque suite de couleurs proposée.

25) Un voyageur de commerce doit se rendre dans N villes. Il part de l'une d'entre elles et doit revenir à son point de départ. Il connaît la distance entre chacune des villes et il désire faire un trajet le plus court possible. Pour cela il adopte la stratégie suivante :
à partir de chaque ville, choisir comme ville étape suivante, la plus proche parmi les villes qu'il n'a pas encore visitées. On suppose donnés : la liste des villes, les distances (en kilomètres) entre chacune d'entre elles et le nom de la ville de départ.
Ecrire un algorithme calculant le parcours du voyageur de commerce conforme à cette stratégie, ainsi que le nombre de kilomètres parcouru.

26) A l'entrée d'un parking, un automobiliste retire un ticket sur lequel est indiquée son heure d'arrivée. Avant de quitter le parking, il introduit son ticket dans un appareil qui lui indique la somme à payer et, une fois cette somme acquittée, lui rend la monnaie.
Ecrire un programme qui réalise le travail de l'appareil.
On suppose que :
la durée de stationnement est toujours inférieure à 24 heures.
l'appareil n'accepte que des pièces de 10 F, 5 F, 2 F, 1 F, 1/2 F.
les tarifs de stationnement sont définis par tranches horaires

27) Un wagon comporte 60 places assises dont 30 places non fumeur numérotées de 1 à 30, et 30 places fumeur numérotées de 31 à 60. Ecrire un algorithme permettant de faire la réservation des places du wagon et d'arrêter la réservation lorsque, soit il n'y a plus aucune place assise libre dans le wagon, soit le départ du train est imminent.

28) Ecrire un algorithme qui permette de dire si un mot donné est un mot palindrome, c'est-à-dire identique à son mot miroir.
Exemples : KAYAK peut se lire de droite à gauche ou de gauche à droite

29) Le meneur de jeu propose un mot qui reste caché, mais dont la longueur est donnée. Le jeu consiste à trouver le mot en moins de n coups perdants. Au début, la première lettre du mot est écrite suivie de tirets, pour remplacer les lettres manquantes. Lorsque le joueur propose une lettre, si celle-ci se trouve dans le mot caché, le meneur de jeu remplace les tirets par la lettre proposée à chacune des places où elle apparaît dans le mot caché, sinon il comptabilise ce coup comme perdant. Ecrire un algorithme qui permette de jouer au jeu du pendu.

30) On lit un texte composé de phrases. Chaque phrase se termine par un point suivi d'un blanc. Devant chaque mot d'une phrase il y a au moins un caractère blanc, sauf devant le premier mot du texte. On suppose que les signes de ponctuation sont collés aux mots les précédant.

- 1. Compter le nombre de phrases
- 2. Indiquer combien chaque phrase comporte de mots

31) Compter le nombre d'occurrences du mot PROGRAMME dans un texte donné

32) N boules bleues, rouges ou blanches sont disposées dans N cases en ordre quelconque. Ecrire un algorithme qui range les boules en mettant d'abord toutes les bleues, puis toutes les blanches, puis toutes les rouges

Correction des exercices

1) Ecrire la permutation de deux nombres

```
Ecrire "permutation de deux nombres"
Ecrire "donner deux nombres"
lire x,y
ech = x
x = y
y = ech
ecrire "après permutation x et y valent", x, y
```

2) Permutation circulaire de trois nombres

```
Ecrire "Permutation circulaire de trois nombres"
lire x, y, z
ech = x
x = y
y = z
z = ech
ecrire "après permutation ", x, y, z
```

3) Composer un algorithme qui lit trois noms frappés successivement au clavier d'un poste de travail puis affiche à l'écran ces trois noms, dans l'ordre alphabétique

```
ecrire "taper un nom"; lire nom1
ecrire "encore un"; lire nom2
ecrire "encore un"; lire nom3
si ( nom1 < nom2 ) et ( nom1 < nom3 )
        alors   ecrire nom1
                si nom2 < nom3
                        alors   ecrire nom2, nom3
                        sinon   ecrire nom3, nom2
                fin si
        sinon
                si nom2 < nom3
                        alors   ecrire nom2
                                si nom1 < nom3
                                        alors   ecrire nom1, nom3
                                        sinon   ecrire nom3, nom1
                                fin si
                        sinon   afficher nom3
                                si nom1 < nom2
                                        alors   ecrire nom1, nom2
                                        sinon   ecrire nom2, nom1
                                fin si
                fin si
fin si
```

4) Trois valeurs numériques ont été lues dans trois variables numériques x, y, et Z. Composez une séquence qui calcule, dans une variable numérique C, le nombre de valeurs distinctes (3, 2, ou 1)

```
si ( x = y ) et ( y = z )
        alors   c = 1
        sinon
                si ( x ≠ y ) et ( y ≠ z ) et ( x ≠ z )
                        alors   c = 3
                        sinon   c = 2
                fin si
fin si
```

6) On a relevé sur une feuille une liste comportant 40 noms. Composez un algorithme qui effectue la saisie de ces noms, dans l'ordre dans lequel ils apparaissent sur cette liste, puis affiche ces noms à l'écran, dans l'ordre inverse.

```
i = 0
faire
        lire nom
        sortir si nom = 0
        tab[i] = nom
        i = i + 1
tant que nom <> 0
pour j de i à 1
faire
        j = j - 1
        ecrire tab[j]
fin faire
```

7) On reprend la liste de noms. Composez un algorithme qui saisit et affiche les noms, dans l'ordre dans lequel ils apparaissent sur la liste, à raison de deux noms par ligne d'écran. Si le nombre de noms est impair, la dernière ligne ne contiendra qu'un nom.

```
itérer
        lire nom1
        sortir si nom1 = 0
        lire nom2
        sortir si nom2 = 0
        ecrire nom1 nom2
fin itérer
si nom1 <> 0
        alors   ecrire nom1
fin si
```

Exercice 24

Algorithme *générale*

Initialisations
Itérer
    Jouer une partie
    Demander si le joueur veut continuer
    Sortir si le joueur répond "non"
fin itérer

Algorithme *Jouer-une -partie*

début
    Mettre à zéro les scores de la partie
    Donner deux cartes au joueur en les montrant
    Calculer son score
    Donner deux cartes au banquier sans les montrer
    Calculer son score
    Laisser jouer le joueur en lui montrant ses cartes
    Faire jouer le banquier en montrant les cartes
    Comparer les scores et annoncer le gagnant
fin

Algorithme *donner-une-carte* (participant , montrer)

début
    si jeu épuisé
        alors mélanger
    fin si
    valeur ← carte suivante
    ajouter point au score (participant)
    si montrer
        alors afficher carte
    fin si
    si valeur = as
        alors nb_as (participant) ← nb_as (participant) + 1
    fin si
fin

Algorithme *je-suis-le-joueur*

```
début
    itérer
        demander si le joueur veut une carte
        si (le joueur répond "non" ) ou (son score dépasse 21)
            alors  sortir
            sinon  donner une carte ( joueur , vrai )
        fin si
    fin itérer
fin
```

Algorithme *je-suis-le-banquier*

```
début
    itérer
        demander si le joueur veut une carte
        si score(banquier) > score(joueur)
            alors  sortir
            sinon  donner une carte ( banquier , vrai )
        fin si
    fin itérer
fin
```

Exercice 25

Algorithme *général*

lire le nombre BN de villes ; lire le nom des villes ; lire les distances entre villes
initialiser la distance parcourue à 0
initialiser le tableau VISIT à FAUX
lire le nom de la ville de départ
écrire le nom de la ville de départ
calculer le numéro de la ville de départ
la ville de départ est visitée
la ville de départ devient la ville courante
pour J **variant de** 2 **à** N **répéter**
    chercher la ville non visitée la plus proche de la ville courante
    ajouter à la distance parcourue, entre la ville courante et la ville la plus proche
    la ville la plus proche devient visitée
    la ville la plus proche devient la ville courante
    écrire le nom de la ville courante
fin pour
ajouter à la distance parcourue, la distance entre la dernière ville et la ville de départ
écrire la ville de départ
écrire la distance parcourue

Algorithme *général*

```
ENTIER :    J, N, VCOUR, VPRO, VDEP, tab DIST [20, 20]
CHAINE :    DEP, tab VILLE [20]
BOOLEEN :       tab VISIT [20]
    LECTURE ( N , VILLE , DIST )
    P ← 0
    pour J variant de 1 à N répéter VISIT [ J ] ← FAUX  fin pour
    lire DEP
    VDEP ← 0
    répéter  VDEP ← VDEP + 1 jusqu'à VILLE [ VDEP ] = DEP
    VISIT [ VDEP ] ← VRAI
    écrire DEP
    VCOUR ← VDEP
    pour J variant de 2 à N répéter
        VPRO ← PROCHE ( VCOUR , N , VISIT )
        P ← P + DIST [ VCOUR , VPRO ]
        VISIT [ VPRO ] ← VRAI
        VCOUR ← VPRO
        écrire VILLE [ VCOUR ]
    fin pour
    P ← P + DIST ( VCOUR , VDEP )
    écrire DEP ; écrire P
```

```
Algorithme lecture ( N , VILLE , DIST )
ENTIER :   T, tab DIST [20, 20]
STRING :   tab VILLE [20]
ENTIER :   j, k
    lire N
    pour J variant de 1 à N répéter
        lire VILLE [ J ]
        DIST [ J, J ] ← 0
        K ← 1
        tant que K < J répéter
            écrire « donner la distance entre VILLE [ K ] et VILLE [ J ] »
            lire DIST [ K , J ]
            DIST [ J , K ] ← DIST [ K , J ]
        fin tant que
    fin pour
```

```
Algorithme PROCHE ( P , N , VISIT )
ENTIER :   P, N
BOOLEEN :      tab VISIT [ 20 ]
ENTIER :   J, MIN :
    MIN ← 10000
    pour J variant de 1 à N répéter
        si ( VISIT [ J ] = FAUX ) et ( DIST [ P, J ] < MIN )
            alors MIN ← DIST [ P , J ]
        fin si
    fin pour
    retour MIN
```

Exercice 26

Algorithme *général*

Lecture du nombre de tranches horaires
Lecture des bornes supérieures des tranches horaires
Lecture du tarif de chaque tranche
Lecture du nombre de pièces de chaque sorte mises initialement dans l'appareil
répéter
    Lire l'heure d'arrivée de l'automobiliste
    Lire l'heure de départ de l'automobiliste
    Calculer la durée du stationnement
    Calculer le tarif du stationnement
    Ecrire la somme due par l'automobiliste
    Lire le nombre de pièces de chaque sorte fourni en paiement par l'automobiliste
    si la somme payée n'est pas suffisante
        alors
            Rendre ses pièces à l'automobiliste
            Ecrire « paiement insuffisant »
            Rendre son ticket à l'automobiliste
        sinon
            Ajouter les pièces fournies par l'automobiliste à celles de l'appareil
            Rendre la monnaie
            Calculer le nombre de pièces de chaque sorte restant dans l'appareil
            Ecrire « payé » sur le ticket
            Rendre le ticket à l'automobiliste
    fin si
jusqu'à arrêt de la machine

Exercice 27

Algorithme *général*

Pour j **variant de** 1 **à** 60 **répéter** W [j] ←FAUX **fin pour**
Initialisation à 0 du nombre de places non fumeur déjà réservées
Initialisation à 0 du nombre de places fumeur déjà réservées
répéter
    Ecrire « voulez-vous une place fumeur (F), non fumeur (NF), ou arrêter la réservation (AR)
    Lire la réponse
    si la réponse est NF
        alors
        si le nombre de places non fumeur déjà réservées est inférieur à 30
            alors
                Le nombre de places non fumeur réservées augmente de 1
                Le numéro de la place réservée est égal au nombre de places non fumeur déjà réservées
                La valeur de W pour ce numéro de place devient VRAI
            sinon écrire : plus de places non fumeur
        fin si
    fin si
    si la réponse est F
        alors
        si le nombre de places fumeur déjà réservées est inférieur à 30
            alors Le nombre de places fumeur réservées augmente de 1
                Le numéro de la place réservée est égal au nombre de places fumeur déjà réservées
                La valeur de W pour ce numéro de place devient VRAI
            sinon écrire : plus de places non fumeur
        fin si
    fin si
jusqu'à ((plus de places non fumeur) et (plus de places fumeur)) ou (la réponse est AR))
écrire le tableau W

Algorithme *général*

```
ENTIER :    J, K, N
BOOLEEN:    tab W [60]
CHAINE :    var REP

pour N variant de 1 à 60 répéter W(N) ← FAUX fin pour
J ← 0; K← 0
répéter
        écrire "voulez-vous une place non fumeur(NF), fumeur(F), ou arrêter la
        réservation(AR) ? "
        lire REP
        au cas où :
                REP = 'NF':
                        si J < 30
                                alors J ← J + 1
                                      W [ J ] ← VRAI
                                      écrire J
                                sinon       écrire plus de places non fumeur
                        fin si
                REP='F'
                        si K < 30
                                alors K ← K + 1
                                      W [ K+30 ] ← VRAI
                                      écrire K + 30
                                sinon       écrire plus de places fumeur
                        fin si
jusqu'à ( (J = 30) et (K = 30) ) ou (REP = 'AR')
pour N variant de 1 à 60 répéter
        si W [ N ] = VRAI
                alors écrire la place N est réservée
                sinon       écrire la place N est libre
        fin si
fin pour
```

Exercice 28

Algorithme *général*

Lecture du mot donné et calcul de sa longueur L
initialisation du compteur de boucle J à 1
répéter
    si la J-ième et la (L-J+1)-ième lettres du mot sont égales
        alors passer au couple suivant, c'est-à-dire augmenter le compteur de J + 1
        sinon sortir de la boucle (Le mot n'est pas un palindrome)
jusqu'à (J > L / 2) ou (indication de sortie de boucle)
si J > L / 2
    alors écrire "le mot est palindrome"
    sinon écrire "le mot n'est pas un palindrome"
fin si

Algorithme *général*

```
ENTIER:      J, L
BOOLEEN :    STOP
CARACTERE tab mot [ 25 ]
```

L ← 0
répéter L ← L + 1
    lire MOT [L]
jusqu'à MOT [L] = Marque_de_fin_de_chaine
L ← L - 1
STOP ← VRAI
J ← 1
répéter
    si MOT(J) = MOT(L - J + 1)
        alors J ← J + 1
        sinon STOP ← FAUX
jusqu'à (J > L / 2) ou (STOP = FAUX)
si STOP = VRAI
    alors écrire "le mot est un palindrome"
    sinon écrire "le mot n'est pas un palindrome"
fin si

Exercice 29

Algorithme *général*

Lire le mot caché et calculer sa longueur
Lire le nombre maximum de coups perdants autorisé
Initialiser le mot trouvé par un mot formé de la première lettre du mot caché suivi par des tirets
Initialiser le nombre de coups perdants à 0
Ecrire l'état initial du mot trouvé
répéter
    Ecrire « proposer une lettre »
    Lire cette lettre
    si cette lettre se trouve dans le mot caché
        alors Trouver toutes les occurrences de cette lettre dans le mot caché et remplacer les tirets du mot trouvé par la lettre proposée à chacun des emplacements trouvés précédemment
        sinon Augmenter le nombre de coups perdants de 1
    fin si
    Ecrire le mot trouvé
jusqu'à (le mot trouvé et le mot caché sont égaux) ou
    (le nombre de coups perdants est égal au nombre maximum de coups perdant autorisé)
si le mot caché est égal au mot trouvé
    alors Ecrire « vous avez gagné »
    sinon Ecrire « vous avez perdu »
fin si

Algorithme *général*

```
ENTIER :        J , K, L, N
CARACTERE :     LETT
CAR :           tab MCACH [25], TROUV [25]
BOOLEEN :       VERIF
   MCACH   : mot trouvé (donné par le meneur de jeu)
   TROUV   : mot trouvé
   L       : longueur du mot caché
   N       : nombre maximum de coups perdants autorisé
   J       : compteur du nombre de coups perdants
   K       : compteur d'itérations
   LETT    : lettre proposée par le joueur
   VERIF   : variable booléenne
      L ← 0
      répéter L ← L + 1
            lire MCACH [ L ]
      jusqu'à MCACH [ L ] = '$'
      L ← L - 1; lire N
      TROUV [1] ← MCACH [1]
      pour K variant de 2 à L répéter TROUV [ K ] ← '-' fin pour
      J ← 0
      écrire TROUV
      répéter écrire « proposer une lettre »
            lire LEFT; VERIF ← FAUX
            pour K variant de 2 à L répéter
                  si MCACH [K] = LEFT
                        alors  TROUV [K] ← LEFT
                               VERIF ← VRAI
                  fin si
            fin pour
            si VERIF = FAUX
                  alors  J ← J + 1
            écrire TROUV
      jusqu'à ( MCACH = TROUV ) ou ( J = N )
      si J < N
            alors  écrire "vous avez gagné en " J " coups"
            sinon écrire "vous avez perdu"
      fin si
      écrire MCACH
```

Exercice 30

Algorithme *général*

initialiser le compteur du nombre de phrases à 0
initialiser le compteur du nombre de mots de la première phrase à 0
lire le premier caractère du texte
répéter
au cas où :
premier cas (le caractère lu est un blanc) :
Augmenter de 1 le compteur de mots
répéter lire le caractère suivant
jusqu'à ce que le caractère lu ne soit pas un blanc
deuxième cas (le caractère lu est un point) :
Lire le caractère suivant
si ce caractère est un blanc ou un $
alors augmenter de 1 le compteur de phrases
augmenter de 1 le compteur de mots
écrire le numéro de la phrase et son nombre de mots
reinitialiser le compteur de mots
tant que le caractère lu est un blanc répéter
lire le caractère suivant
fin tant que
fin si
troisième cas (le caractère n'est ni un blanc, ni un point :
lire le caractère suivant
jusqu'à le caractère lu est un $

Algorithme *général*

```
ENTIER :        M, N
CARACTERE :     A
```

M← 0; N ← 0; lire A
répéter
au cas ou
A = ' ': M ← M + 1
répéter lire A **jusqu'à** A ≠ ' ' **fin tant que**
A = '.': lire A
si A = ' ' ou A = '$'
alors
N ← N + 1 ; M ← M + 1 ; écrire N, M ; M ← 0
tant que A = ' ' **répéter** lire A **fin tant que**
fin si
A ≠' ' et A ≠ '.': lire A
jusqu'à A = '$'
écrire N

Exercice 31

Plan de l'algorithme

Initialiser le compteur du nombre d'occurrences du mot PROGRAMME à 0
Lecture du mot PROGRAMME et calcul de sa longueur
lire le premier caractère du texte
tant que le caractère lu est un blanc r**épéte**r lire un caractère **fin tant que**
répéter
    examiner si le mot du texte est ou non le mot PROGRAMME
    si oui augmenter le compteur du nombre d'occurrences de 1
    se pointer sur la première lettre du mot suivant
jusqu'à fin du texte
écrire la valeur du compteur

Algorithme

```
MOT          : tableau de caractères contenant le mot « programme »
LM           : longueur du mot « programme »
K            : pointeur dans le tableau MOT
NP           : compteur du nombre d'occurrences du mot « programme »
A            : mémoire contenant successivement tous les caractères du texte
ENTIER :          K, LM, NP
CARACTERE :       A ,  tab MOT [15]
    NP ← 0
    LM ← 0
    répéter LM ← LM + 1 ; lire MOT [ LM ] jusqu'à MOT [ LM ] = '$'
    LM ← LM - 1
    lire A
    tant que A = ' ' répéter lire A fin tant que
    répéter K ← 1
        tant que ( MOT [ K ] = A ) et ( K <= LM )
            répéter     K← K + 1 ; lire A
        fin tant que
/* K = LM + 1 signifie qu'on a trouvé les LM lettres du mot programme dans le texte */
        si K = LM + 1
            alors si (A = ' ') ou (A = ',') ou (A = ';') ou (A = ',')
                        alors NP ← NP + 1
                  fin si
        fin si
        tant que ( A ≠ '$' ) et ( A ≠ ' ' ) répéter lire A fin tant que
/*si A n'était pas un blanc, il fallait finir le mot, donc trouver un blanc */
        si A ≠ '$'
            alors
                  tant que A = ' ' répéter lire A fin tant que
/*on est alors pointé sur la première lettre du mot suivant */
    jusqu'à A = '$'
    écrire NP
```

Exercice 32

Plan de l'algorithme

lire le nombre N de boules
lire la position initiale des boules
pour J **variant** de 1 à N **répéter**
  si la boule est bleue
    alors intercaler cette boule derrière la dernière boule bleue déjà rangée dans le tableau
  fin si
  si la boule est rouge
    alors intercaler cette boule devant la première boule rouge déjà rangée dans le tableau
  fin si
écrire la position finale des boules

Algorithme

```
N       : nombre de boules à ranger
J       : compteur du nombre de boules examinées
K       : position dans le tableau de la boule à examiner
PB      : rang de la dernière boule bleue rangée
PR      : rang de la dernière boule rouge rangée
A       : tableau de caractères représentant la suite des boules
ENTIER :            J, K, PB, PR, N
CARACTERE :         tab A [ 100 ]
    lire N
    lire A
    PB ← 0; PR ← N + 1
    K ← 1
    pour J variant 1 à N répéter
        au cas où :
                A(K) = 'BLEU' :
                        intercaler le K-ième élément du tableau A à la (PB + 1)-ième place du tableau
                        K ← K + 1
                        PB ← PB + 1
                A(K) = 'BLANC'
                        K ← K + 1
                A(K) = 'ROUGE'
                        intercaler le K-ième élément du tableau A à la (PR-1)-ième place du tableau
                        PR ← PR - 1
    fin pour
    Pour J variant de 1 à N répéter écrire A(J) fin pour
```

Table des matières

Analyse et résolution de problèmes

(de la programmation structurée à la programmation orientée objets)

Pierre Lefebvre

Gréta tertiaire Sud 93 du Raincy