U22 – ALGORITHMIQUE APPLIQUÉE

Les thématiques abordées lors de l'étude de ce module sont très ouvertes, mais l'objectif visé, à l'intérieur du processus de conception, est fortement ciblé. On veillera à ce que les situations proposées soient mathématiquement achevées. Alors qu'une solution, voire un pré-algorithme, peuvent être décrits de manière très libre, textuelle ou graphique, par formules ou symboles, par l'exemple ou de manière inachevée, on s'attache ici à l'exprimer en utilisant les outils algorithmiques usuels, pour la rendre directement convertible et exécutable sur machine.

Afin de faciliter la compréhension des mécanismes et la détection d'éventuelles erreurs, il est impératif de les concrétiser par l'emploi d'un langage de programmation et de conduire l'étudiant à réaliser des tests. La tâche inverse, consistant à comprendre un algorithme, présente également un grand intérêt pour l'assimilation des mécanismes et lors d'opérations de maintenance.

Les compétences et savoir-faire à acquérir concernent la compréhension des solutions proposées, l'interprétation des algorithmes (découverte de leurs rôles), leur construction conforme aux prescriptions et convenablement documentée, leur transcription dans un langage informatique, leur mise au point et leur validation.

Les contrôles d'exécution constituent le cœur des mécanismes algorithmiques de base. À ce titre, on attachera un soin tout particulier à leur étude progressive mais détaillée. On ne se limitera, en aucun cas, à en définir les fonctionnements. Leur maîtrise pratique est essentielle et les évaluations doivent être centrées sur eux.

Pour l'écriture des algorithmes, on évitera l'utilisation de symboles graphiques contraignants. Une représentation textuelle convenablement indentée avec des indicateurs de début et de fin explicites conviendra. Pour faciliter la compréhension, on exigera également un en-tête composé d'un nom, d'un rôle, de l'indication des données d'entrée et de sortie et de la déclaration typée des données locales. Des commentaires seront ajoutés, si utiles, notamment pour préciser le rôle des variables et d'éventuelles indications méthodologiques.

Pour la programmation, on peut notamment utiliser un tableur, un langage de calcul formel ou un langage de haut niveau, éventuellement exécutable dans un navigateur. Aucun langage ni aucun logiciel n'est imposé, mais il convient de s'assurer qu'il est accepté par le centre d'examen.

Les sujets empruntés à la vie courante pourront être utilisés à chaque fois qu'ils permettent d'illustrer un mécanisme simple avec pertinence. Sinon, on préférera utiliser des sujets dérivés directement des modules mathématiques de l'U21 et, avec un certain équilibre, des sujets associés à des thématiques informatiques (par exemple : codage, cryptage et décryptage, redondance de sécurité, tri itératif et tri récursif, option de graphes). Ces sujets seront abordés à fin d'illustration des concepts fondamentaux d'algorithmique et de familiarisation avec les notions, les entités et les méthodes manipulés par ces thèmes, sans qu'aucune connaissance spécifique ne soit exigible de l'étudiant dans ces derniers domaines.

Généralités

Les concepts fondamentaux (algorithme, finitude, modularité, identifiant, constante, variable, fonction, procédure, expression numérique, expression conditionnelle et plus généralement booléenne…) seront acquis par l'usage, sans faire appel à des définitions formelles préalables.	Le formalisme a posteriori, utile à une compréhension fine, n'est pas exclu, mais ne peut faire l'objet d'évaluation.

Données manipulées

Types simples : entier naturel, entier relatif, réel, booléen,	Peu importe la façon dont les valeurs de vérité sont représentées.
chaîne de caractères.	On évitera de considérer une chaîne comme un tableau de caractères.
Tableaux à une ou deux dimensions de type homogène, tableaux à deux dimensions constitués de tableaux à une dimension dont les types ne sont pas homogènes.	On adaptera la construction et l'exploitation de ces tableaux aux possibilités de l'outil informatique utilisé. Les structures de données, les objets, les fichiers ne sont pas au programme de mathématiques : ils figurent dans les enseignements professionnels.
Paramètres d'entrée, valeur(s) retournée(s) par une fonction, variables globales ou locales.	Sans aborder la programmation objet, les concepts de modularité, d'interface et de portée des variables devront être assimilés.

Instructions de base et opérateurs utilisés

Lecture, écriture. Affectation, affectation récursive (la variable affectée participe à l'expression évaluée). Opérateurs numériques : addition, soustraction, multiplication, division, exponentiation, quotient et reste de la division entière, signes. Fonctions mathématiques usuelles.	
Opérateurs de comparaison : =, <> ou !=, <, <=, >, >=	Entre numériques et entre chaînes de caractères.
Opérateurs booléens : non, et, ou, oux Opérateurs booléens bit à bit	On interprétera notamment en termes de masque, de mise à un, de mise à zéro, de changement d'état.
Opérateurs de chaînes : concaténation. Fonctions permettant l'extraction en début, milieu ou fin ; la recherche d'un motif. Transtypage	L'usage d'expressions régulières simples est possible, mais l'étude des expressions régulières est hors programme.
Toute autre instruction, fonction ou procédure utile aux algorithmes traités.	Les descriptions sémantique et syntaxique précises seront alors mises à disposition de l'étudiant.

Contrôle d'exécution

Par défaut : l'exécution séquentielle. Exécution à structure conditionnelle (si-alors-sinon),	Afin d'en maîtriser le fonctionnement, les structures d'exécution seront elles-mêmes présentées sous forme d'algorithmes.
Exécution à structure itérative déterministe (pour) et indéterministe (tant que / répéter jusqu'à ce que). Méthodologie de construction des structures itératives : raisonnement par récurrence, initialisation, mise à jour itérative et neutralisation des cas indésirables, calcul itératif (souvent récursif), mise en forme finale.	Le raisonnement par récurrence n'a pas à être évalué pour des démonstrations. Il est introduit pour servir de base à une méthodologie non empirique de construction des itérations.

Symboles $\sum_{i=\cdots}^{\cdots}\cdots$ et $\prod_{i=\cdots}^{\cdots}\cdots$, traduction algorithmique.	
Structures imbriquées, y compris quand les éléments de contrôle des structures internes dépendent de ceux des structures externes. Le nombre d'imbrications n'est pas limité, sauf pour les itérations en dépendance, où on se limitera à deux.	On évitera les excès de complexité.
Récursivité. Nécessité d'un test. Nécessité de cas particuliers résolus sans appel à la récursivité, finitude.	Pas de récursivité mutuelle de plusieurs procédures. Aucune connaissance théorique sur la complexité ou la conversion en itérations.

Travaux pratiques

1° Exemple(s) de récursivité terminale, conversion en algorithme non récursif. Exemple de récursivité non terminale.	Aucune connaissance n'est exigible.
2° Variantes où les notions de complexité temporelle et de complexité spatiale peuvent être abordées.	L'étude de la complexité des algorithmes se limitera au calcul d'un nombre minimum ou maximum d'opérations significatives ou de taille des données.
3° Exemples d'effets indésirables (effets de bord, évaluation partielle lors de calcul d'expressions booléennes, débordements ou approximations numériques, transtypage, utilisation d'indices hors domaine,…).	Afin de mieux sensibiliser l'étudiant à certains risques, on s'efforcera de présenter des cas aux conséquences spectaculaires. Aucune théorie n'est au programme.
4° Algorithme délibérément erroné dont les défauts seront repérés par débogage.	On procédera essentiellement par suivi de variables.
5° Interprétation d'algorithmes.	L'ajout d'une ou plusieurs procédures ou fonctions à un ensemble interdépendant peut constituer une excellente base.