Université Virtuelle Africaine

INFORMATIQUE APPLIQUÉE: ITI 2200

STRUCTURES DE DONNÉES ET ALGORITHMES

Djamal Abdoul Nasser Seck

Avant-propos

L'Université Virtuelle Africaine (UVA) est fière de participer à accès à l'éducation dans les pays africains en produisant du matériel d'apprentissage de qualité. Nous sommes également fiers de contribuer à la connaissance globale, pour nos ressources éducatives sont principalement accessibles de l'extérieur du continent africain.

Ce module a été développé dans le cadre d'un programme de diplôme et diplôme en informatique appliquée, en collaboration avec 18 institutions partenaires dans 16 pays africains. Un total de 156 modules ont été développés ou traduits pour assurer la disponibilité en anglais, français et portugais. Ces modules sont également disponibles en tant que ressources éducatives ouvertes (OER) à oer.avu.org.

Au nom de l'Université Virtuelle Africaine et notre patron, nos institutions partenaires, la Banque africaine de développement, je vous invite à utiliser ce module dans votre établissement, pour leur propre éducation, partager aussi largement que possible et participer activement aux communautés AVU de pratique d'intérêt. Nous nous engageons à être à l'avant-garde du développement et de partage ouvert de ressources pédagogiques.

L'Université Virtuelle Africaine (UVA) est une organisation intergouvernementale panafricaine mis en place par lettre recommandée avec un mandat d'augmenter l'accès à l'enseignement supérieur et de formation de qualité grâce à l'utilisation novatrice des technologies de communication de l'information. Une charte instituant la UVA Organisation intergouvernementale, signée à ce jour par dix-neuf (19) Les gouvernements africains - Kenya, Sénégal, Mauritanie, Mali, Côte d'Ivoire, Tanzanie, Mozambique, République démocratique du Congo, Bénin, Ghana, République de Guinée, le Burkina Faso, le Niger, le Soudan du Sud, le Soudan, la Gambie, la Guinée-Bissau, l'Ethiopie et le Cap-Vert.

Les institutions suivantes ont participé au programme informatique appliquée: (1) Université d'Abomey Calavi au Bénin; (2) University of Ougagadougou au Burkina Faso; (3) Université Lumière Bujumbura Burundi; (4) Université de Douala au Cameroun; (5) Université de Nouakchott en Mauritanie; (6) Université Gaston Berger Sénégal; (7) Université des Sciences, Techniques et Technologies de Bamako au Mali (8) Institut de la gestion et de l'administration publique du Ghana; (9) Université des sciences et de la technologie Kwame Nkrumah au Ghana; (10) Université Kenyatta au Kenya; (11) Université Egerton au Kenya; (12) Université d'Addis-Abeba en Ethiopie (13) Université du Rwanda; (14) University of Salaam en Tanzanie Dar; (15) Université Abdou Moumouni Niamey Niger; (16) Université Cheikh Anta Diop au Sénégal; (17) Université pédagogique au Mozambique; E (18) L'Université de la Gambie en Gambie.

Bakary Diallo

le Recteur

Université Virtuelle Africaine

Crédits de production

Auteur

Djamal Nasser

Pair Réviseur

Boune Mohammed

UVA – Coordination Académique

Dr. Marilena Cabral

Coordinateur global Sciences Informatiques Apliquées

Prof Tim Mwololo Waema

Coordinateur du module

Jules Degila

Concepteurs pédagogiques

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Equipe Média

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Droits d'auteur

Supporté par

Projet Multinational II de l'UVA financé par la Banque africaine de développement.

Table Des Matièress

Aperçu du cours

Bienvenue à Structures de données et Algorithmes

Ce cours initie les apprenants à différents types de structures de données et d'algorithmes, à la façon dont des structures de données peuvent être créées et utilisées. L'application des structures de données sera présentée sur la base de différents algorithmes.

Prérequis

- Introduction à l'informatique appliquée
- Principes de la programmation

Matériaux

Les matériaux nécessaires pour compléter ce cours comprennent:

- Manuels d'algorithme, notamment :
- Algorithmes en C : Cours et exercices, Robert Sedgewick, Dunod, 2005
- Introduction à l'algorithmique, Cormen, Leiserson, Rivest et Stein, Dunod, 3e édition, 2010.
- Algorithmes en Pascal et en langage C, Granjon, Dunod, 2004.
- Initiation à l'algorithmique et à la programmation en C, Rémy Malgouyres, Rita Zrour et Fabien Feschet, Dunod, 3e édition, 2015
- Algorithmique - Applications en C, Jean-Michel Léry, Pearson Education, 2005.
- Algorithmique en C, Jean-Michel Léry, Pearson Education, 2e édition, 2010
- Liens vers des ressources en ligne
- Notes de cours
- Ordinateur et connectivité Internet

Objectifs du cours

À la fin de ce cours, l'étudiant devrait être en mesure de

- connaitre les principales structures de données (listes, piles, files, arbre, ensembles, cartes, tas, graphes, etc.)
- Comprendre l'importance des algorithmes dans la résolution des problèmes et déterminer les structures de données les plus efficaces en rapport avec les divers scénarios et problèmes

- Analyser avec compétence le temps d'exécution des divers algorithmes associés aux structures de données
- Créer, modifier, effacer et combiner des structures de données
- Choisir et appliquer la structure de données appropriée pour modéliser un problème donné

Unités

Unité 0: Évaluation diagnostique

Dans cette unité, vous allez découvrir les notions de base en Informatique et les principes de la programmation.

Unité 1 Solution algorithmique des problèmes

Cette unité vous introduit aux concepts de base en définissant la notion d'algorithme et sa relation avec la programmation. Elle vous donne les connaissances nécessaires pour écrire des algorithmes pour résoudre des problèmes.

Unité 2: La récursivité

Dans cette unité, vous avez une introduction à la récursivité qui est le fait qu'un traitement dans un algorithme s'exécute de manière récurrente en faisant appel à lui-même directement ou indirectement. Pour cela, la notion de fonction est abordée pour expliquer comment on peut définir un traitement qu'on peut exécuter plusieurs fois dans un algorithme sans être obligé de réécrire les instructions qui le composent.

Unité 3: Les structures de données et les types abstraits de données

Dans cette unité, il vous est présenté les structures de données de base qu'on peut utiliser pour organiser les données dans un algorithme. Elle présente également les types abstraits qu'on peut définir à partir de ces structures de données en spécifiant des propriétés et des opérations qui leur sont applicables.

Unité 4: Algorithmes de recherche et de tri

Cette unité vous présente les algorithmes de recherche et de tri et vous donne leurs principes de fonctionnement.

Évaluation

Les évaluations formatives (vérification de progrès) sont incluses dans chaque unité.

Les évaluations sommatives (tests et travaux finaux) sont fournies à la fin de chaque module et traitent des connaissances et compétences du module.

Les évaluations sommatives sont gérées à la discrétion de l'établissement qui offre le cours. Le plan d'évaluation proposé est le suivant :

1	Contrôle continu	30%
2	Examen final	70%

Plan

Unité	Sujets et Activités	Durée estimée
0	0 Évaluation diagnostique 1. Informatique de base 2. Principes de la programmation	3 heures
1	Solution algorithmique des problèmes 1. Introduction à la notion d'algorithme 2. Représentation d'un algorithme 3. Caractéristiques d'un algorithme	6 heures
2	La récursivité 1. Présentation de la notion de fonction 2. Présentation de la récursivité	6 heures

3	Les structures de données et les types abstraits de données 1. Notions de structures de données et de types abstraits de données 2. Présentation de structures de données de base 3. Présentation de types abstraits de données	15 heures
4	Algorithmes de recherche et de tri 1. Définition des taches de recherche et de tri 2. Présentation d'algorithmes de recherche 3. Présentation d'algorithmes de tri	9 heures

Lectures et autres ressources

Les lectures et autres ressources dans ce cours sont indiquées ci-dessous.

Unité 0

Lectures et autres ressources obligatoires:

- Paolo Coletti, Basic Computer course book, 8TH Edition: 2014
- Bjarne Stroustrup, Programming: Principles and Practice Using C++, Addison Wesley, 2014

Unité 1

Lectures et autres ressources obligatoires:

- Algorithmique en C, Jean-Michel Léry, Pearson Education, 2e édition, 2010
- Algorithmes en Pascal et en langage C, Granjon, Dunod, 2004.

Lectures et autres ressources optionnelles:

- Algorithmique - Applications en C, Jean-Michel Léry, Pearson Education, 2005.
- Initiation à l'algorithmique et à la programmation en C, Rémy Malgouyres, Rita Zrour et Fabien Feschet, Dunod, 3e édition, 2015

Unité 2

Lectures et autres ressources obligatoires:

- Algorithmes en C : Cours et exercices, Robert Sedgewick, Dunod, 2005
- Introduction à l'algorithmique, Cormen, Leiserson, Rivest et Stein, Dunod, 3e édition, 2010.

Lectures et autres ressources optionnelles:

- Algorithmique en C, Jean-Michel Léry, Pearson Education, 2e édition, 2010
- Algorithmes en Pascal et en langage C, Granjon, Dunod, 2004.
- Initiation à l'algorithmique et à la programmation en C, Rémy Malgouyres, Rita Zrour et Fabien Feschet, Dunod, 3e édition, 2015

Unité 3

Lectures et autres ressources obligatoires:

- Algorithmes en C : Cours et exercices, Robert Sedgewick, Dunod, 2005
- Introduction à l'algorithmique, Cormen, Leiserson, Rivest et Stein, Dunod, 3e édition, 2010.

Lectures et autres ressources optionnelles:

- [Algorithmique en C, Jean-Michel Léry, Pearson Education, 2e édition, 2010
- Algorithmes en Pascal et en langage C, Granjon, Dunod, 2004.
- Initiation à l'algorithmique et à la programmation en C, Rémy Malgouyres, Rita Zrour et Fabien Feschet, Dunod, 3e édition, 2015

Unité 4

Lectures et autres ressources obligatoires:

- Algorithmes en C : Cours et exercices, Robert Sedgewick, Dunod, 2005
- Introduction à l'algorithmique, Cormen, Leiserson, Rivest et Stein, Dunod, 3e édition, 2010.

Lectures et autres ressources optionnelles:

- Algorithmique en C, Jean-Michel Léry, Pearson Education, 2e édition, 2010
- Algorithmes en Pascal et en langage C, Granjon, Dunod, 2004.
- Initiation à l'algorithmique et à la programmation en C, Rémy Malgouyres, Rita Zrour et Fabien Feschet, Dunod, 3e édition, 2015

Unité 0. Évaluation diagnostique

Introduction à l'unité

Cette unité vous permettra de vérifier les connaissances que vous devez avoir avant de commencer le cours. Vous pouvez faire l'évaluation de l'unité avant de faire les activités d'apprentissage pour aider à rafraîchir vos connaissances.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Rappeler les connaissances de base pour la définition et le stockage des données informatiques
- Décrire les principes de base de la programmation
- Identifier les stratégies de résolution de problèmes

TERMES CLÉS

Unité centrale : Partie de l'ordinateur qui contient le processeur, la mémoire centrale, le disque dur et d'autres dispositifs

Entrée : terme désignant soit une entrée ou des changements qui sont insérés dans un système

Sortie : terme désignant un résultat produit système ou un dispositif

Programme : ensemble d'instructions qui indiquent à l'ordinateur ce qu'il faut faire ou comment résoudre une tâche donnée

Système d'exploitation : un programme qui agit comme une interface entre le logiciel et le matériel informatique

Évaluation de l'unité

Vérifiez votre compréhension!

Exercices

Directives

Les apprenants doivent faire des lectures sur les bases informatiques et l'introduction à la programmation qui sont les cours préalables pour ce module.

Ils doivent ensuite répondre aux questions suivantes pour vérifier s'ils ont acquis les connaissances leur permettant de suivre ce module sur les structures de données et les algorithmes.

1. Définir les termes suivants:
    a) ordinateur
    b) programmation
    c) syntaxe
2. Décrire brièvement les termes suivants:
    a) périphériques de stockage
    b) logiciel
    c) matériel
3. Distinguer entre une application et un système d'exploitation.

Système de notation

Les points seront répartis comme suit:

Question	Sous-question	Points
1	a)	2
	b)	2
	c)	2
2	a)	2
	b)	2
	c)	2
3		6
Total		18

Évaluation

1. Définition de termes

    a) ordinateur : un dispositif électronique de traitement de données qui accepte des données d'entrée, les stocke, les traite, et génère des sorties

    b) programmation : est la création d'un ensemble d'instructions pour remplir une tâche spécifique

    c) syntaxe : la structure et les règles d'un langage de programmation

2. Décrire brièvement ce qui suit

    a) périphériques de stockage : dispositifs de stockage (sauvegarde) des données.

    b) logiciel : est un ensemble d'instructions qui dirigent le processeur d'un ordinateur pour effectuer des opérations spécifiques.

    c) Matériel : est la composante physique des ordinateurs

3. Distinguer entre un logiciel d'application et un logiciel de système d'exploitation

Une application est un logiciel constitué d'un ou de plusieurs programmes conçus pour effectuer des tâches spécifiques.

Un système d'exploitation est un logiciel qui gère les ressources matérielles et logicielles de l'ordinateur et fournit des services communs pour les programmes informatiques.

Lectures et autres ressources

Les lectures et ressources de cette unité se trouvent au niveau des lectures et autres ressources du cours.

Unité 1. Solution algorithmique des problèmes

Introduction à l'unité

Cette unité introduit les concepts de base en définissant la notion d'algorithme et sa relation avec la programmation. Elle donne les connaissances nécessaires pour écrire des algorithmes pour résoudre des problèmes.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Décrire un algorithme
- Expliquer la relation entre algorithme et programme
- Décrire les caractéristiques des algorithmes
- Concevoir et écrire des algorithmes en pseudo-code ou avec un organigramme

TERMES CLÉS

Algorithme: séquence finie d'étapes pour accomplir une tâche

Pseudo-code: une description informelle de haut niveau du principe de fonctionnement d'un programme correspondant à un algorithme

Organigramme: représentation schématique d'un algorithme

Données: informations utilisées dans un algorithme ou un programme

Activités d'apprentissage

Activité 1.1 - Lecture sur algorithmes

Introduction

Cette activité porte sur les concepts de base des algorithmes. Après la définition de la notion d'algorithme, on parle de sa représentation ainsi que de ses caractéristiques.

Détails de l'activité

1. **Notion d'algorithme**

Un algorithme est la description d'un ensemble d'actions à effectuer pour résoudre un problème donné.

Il faut distinguer le concepteur de l'algorithme et l'exécutant de l'algorithme.

Le concepteur de l'algorithme est une personne qui connaît la solution du problème à résoudre et la décrit sous la forme d'un ensemble ordonné d'actions à exécuter. Ces actions sont appelées instructions et cette description, comme nous l'avons déjà dit, représente l'algorithme.

L'exécutant est une personne ou une machine (un ordinateur par exemple) qui effectue les actions décrites dans l'algorithme pour aboutir à la solution du problème.

2. Représentation d'un algorithme

La représentation d'un algorithme doit se faire en utilisant une notation simplifiée de manière à ce qu'elle soit compréhensible de tous.

Le langage algorithmique, aussi appelé pseudo-code, est la notation la plus utilisée. C'est un langage proche du langage courant mais qui n'est pas standardisé.

Un algorithme peut aussi être représenté de manière graphique par un organigramme utilisant des symboles graphiques pour décrire la suite d'actions à exécuter.

La représentation graphique est intéressante mais seulement pour des algorithmes de petite taille. Un long algorithme nécessiterait un espace important pour la représentation des symboles graphiques et ne serait ainsi pas très lisible. C'est pourquoi, dans la suite de ce module, nous utiliserons le langage algorithmique pour représenter les algorithmes.

3. Notion de programme

Dans le cas où l'exécutant est un ordinateur, un algorithme doit d'abord être traduit en ce qu'on appelle un programme en utilisant un langage standardisé appelé langage de programmation. Le programme, encore appelé programme source ou code source, contient l'ensemble des instructions que l'ordinateur doit exécuter. Ensuite intervient un troisième acteur appelé utilisateur qui est une personne qui interagit avec l'ordinateur en lui fournissant des informations et en recevant les résultats de l'exécution qui peuvent être des informations affichées à l'écran.

4. Les types de données et les opérations

Un algorithme de résolution d'un problème agit sur des informations également appelées données et qui peuvent être de plusieurs types. Un type permet de caractériser des données de même nature en déterminant les valeurs possibles pour ces informations ainsi que les opérations qu'on peut leur appliquer. On distingue les types de données de base suivants:

- entier (valeurs possibles : les nombres entiers positifs ou négatifs)

- réel (valeurs possibles : les nombres réels positifs ou négatifs)

- caractère (valeurs possibles : lettre, chiffre, ponctuation, espace, etc.)

- chaîne (valeurs possibles : toute suite de zéro, un ou plusieurs caractères)

- booléen (valeurs possibles : VRAI, FAUX)

Dans un algorithme, les opérations qu'on peut appliquer sur les données dépendent de leurs types. On distingue les opérations de base suivantes:

- Les opérations arithmétiques pour des données de type entier ou réel:

addition (+), soustraction (-), multiplication (×), division réelle (/), division entière (div), reste de la division entière ou modulo (mod)

- Les opérations de comparaison pour tous les types de données :

inférieur (<), inférieur ou égal (<=), supérieur (>), supérieur ou égal (>=), égal (= =), différent (≠)

- Les opérations logiques pour des données de type booléen :

ET logique (ET), OU logique (OU), Négation (NON)

5. Les variables et les constantes

Les variables et les constantes servent à contenir les informations utilisées dans un algorithme. Une variable ou une constante est caractérisée par :

- un identificateur, qui est un nom unique qui permet de la désigner ;

- un type, qui correspond au type de l'information qu'elle contient ;

- une valeur, c'est à dire l'information qu'elle contient.

La valeur d'une variable peut être modifiée dans un algorithme contrairement à celle d'une constante. Mais avant de les utiliser dans un algorithme il faut d'abord les déclarer.

Pour déclarer une variable, on choisit son identificateur et son type selon la syntaxe suivante :

id_variable : type Pour déclarer une constante, on choisit son identificateur et son type tout en lui donnant une valeur selon la syntaxe suivante :

id_constante = valeur : type

6. Structure générale d'un algorithme

En pseudo-code, on peut structurer un algorithme de la manière suivante :

Algorithme Nom de l'algorithme

Constante

Déclaration des constantes

Début

Variable

Déclaration des variables

Instructions

Fin

7. Les instructions d'un algorithme

Avec l'instruction d'affectation l'ordinateur donne à une variable la valeur d'une expression. Elle se fait avec l'opérateur = selon la syntaxe suivante :

id_variable = expression

où expression est soit un identificateur de variable ou de constante, soit une valeur, soit une opération, c'est à dire une combinaison d'identificateurs, de valeurs et d'opérateurs.

Avec l'instruction d'écriture, l'ordinateur affiche à l'écran des valeurs des expressions. Sa syntaxe est la suivante:

Ecrire(expression1 , expression2 , … , expressionN)

Avec l'instruction de lecture, l'ordinateur reçoit des valeurs entrées au clavier par l'utilisateur et les affecte à des variables données en paramètre. Sa syntaxe est la suivante:

Lire(id_variable1 , id_variable2 , … , id_variableN)

Il est conseillé de précéder l'instruction de lecture par une instruction d'écriture pour l'affichage d'un message invitant l'utilisateur à entrer une ou plusieurs valeurs.

8. Les structures de contrôle

Les structures de contrôle permettent de déterminer l'enchaînement ou la fréquence d'exécution des instructions d'un algorithme.

La structure Si permet l'exécution d'un groupe d'instructions ou d'une autre en fonction de la valeur d'une ou de plusieurs conditions suivant la syntaxe générale suivante :

Si (condition1) Alors

instructions1

Sinon Si (condition2) Alors

instructions2

...

Sinon Si (conditionN) Alors

instructionsN

Sinon

instructions

FinSi

Une condition est une expression de type booléen. Les opérations de comparaison et les opérations logiques sont des conditions car elles donnent comme résultats VRAI ou FAUX.

Les structures répétitives, encore appelées boucles, permettent de répéter plusieurs fois l'exécution d'un groupe d'instructions.

La boucle Tant Que permet de répéter l'exécution d'un groupe d'instructions tant qu'une condition est vraie. Sa syntaxe est:

Tant Que (condition) Faire

instructions

FinTantQue

La boucle Faire…Tant Que permet d'exécuter un groupe d'instructions au moins une fois et de le répéter tant qu'une condition est vraie. Sa syntaxe est :

Faire

instructions

Tant Que (condition)

La boucle Pour permet d'exécuter un groupe d'instructions pour chaque valeur prise dans un intervalle par une variable de type entier appelée compteur. Sa syntaxe est la suivante:

Pour id_Variable = val_initiale à val_finale par pas de val_pas Faire

instructions

FinPour

Si le pas n'est pas précisé, sa valeur est par défaut égale à 1.

Conclusion

Dans cette activité, la notion d'algorithme a d'abord été définie. Ensuite, des concepts de base nécessaires pour déterminer et représenter des algorithmes ont été abordés.

Évaluation

Répondre aux questions suivantes:

1. Définir ce qu'est un algorithme.
2. Donner un algorithme pour échanger les contenus de deux variables

Résumé de l'unité

Cette unité a introduit les concepts de base en définissant la notion d'algorithme et sa relation avec la programmation. Elle donne les connaissances nécessaires pour écrire des algorithmes afin de résoudre des problèmes: représentation d'un algorithme, types de données utilisées, instructions et structures de contrôle.

Évaluation de l'unité

Vérifiez votre compréhension!

Évaluation sommative

Directives

Répondre aux questions suivantes:

1. Définir ce qu'est un algorithme.
2. Donner un algorithme pour échanger les contenus de deux variables

Système de notation

Les points sont répartis comme suit:

Question	Points
1	4
2	6
Total	10

Évaluation

1. Définir ce qu'est un algorithme

Un algorithme est une suite d'action à effectuer pour résoudre un problème.

2. Donner un algorithme pour échanger les valeurs de deux variables

Soient deux variables x et y.

Si on commence par l'instruction x = y on va perd la valeur de x.

Si on commence par l'instruction y = x on va perdre la valeur de y.

La solution consiste à utiliser une troisième variable pour garder d'abord la valeur de x. Ensuite, on affecte à x la valeur de y. Enfin, on affecte à y la valeur de la troisième qui est celle de x qu'on y avait gardée.

Lectures et autres ressources

Les lectures et autres ressources de cette unité se trouvent au niveau des lectures et autres ressources du cours.

Unité 2. La récursivité

Introduction à l'unité

Cette unité est une introduction à la récursivité qui est le fait qu'un traitement dans un algorithme s'exécute de manière récurrente en faisant appel à lui-même directement ou indirectement. Pour cela, la notion de fonction est d'abord abordée pour expliquer comment on peut définir un traiement qu'on peut exécuter plusieurs fois dans un algorithme sans être obligé de réécrire les instructions qui le composent.

L'apprenant sera en mesure de comprendre la notion de récursivité et de résoudre les problèmes qui sont par nature récursif.

Objectifs de l'unité

À la fin de cette unité, l'apprenant devrait être capable de:

- développer des algorithmes pour les programmes récursifs
- Ecrire des algorithmes récursifs facilement transformables en programmes
- Mettre en œuvre la formulation récursive d'un problème.
- Expliquer la récursivité comme une forme d'itération.

TERMES CLÉS

Fonction: un groupe d"instructions regroupées sous un même nom

Récursivité: le fait qu'un traitement défini comme une fonction se répète plusieurs fois en faisant appel à lui-même

Cas de base: un test qui arrête la suite des appels récursifs

Activités d'apprentissage

Activité 2.1 - Lecture sur les notion de fonction et de récursivité

Introduction

Cette activité introduit d'abord la notion de fonction en expliquant comment on peut l'utiliser dans un algorithme. La notion de récursivité est ensuite abordée en donnant sa définition et son principe de fonctionnement.

Détails de l'activité

1. **Notion de fonction**

Une fonction est un regroupement d'instructions défini dans un algorithme et qu'on peut appeler par son nom à plusieurs endroits pour l'exécution des instructions qu'elle regroupe.

Une fonction peut avoir une valeur de retour qui est un résultat produit à la fin de son exécution Dans ce cas, elle peut être définie avec la syntaxe suivante :

NomFonction(param1: type1, param2: type2,…, paramN: typeN) : type

Début

variable

déclaration de variables

instructions

…

Retourner expression

FinFonction

Les paramètres param1, param2, …, paramN figurant dans l'entête de la définition de la fonction sont appelés paramètres formels.

Le type de la fonction est celui de sa valeur de retour qui est la valeur d'une expression donnée par la fonction avec l'instruction Retourner.

Si la fonction n'a pas de valeur de retour, elle n'a alors pas de type et sa définition se fait avec la syntaxe suivante :

NomFonction(param1: type1, param2: type2,…, paramN: typeN)

Début

variable

déclaration de variables

instructions

FinFonction

Une fonction sans valeur de retour est aussi appelée une procédure.

L'appel d'une fonction se fait par son nom suivi, entre parenthèses, de paramètres appelés paramètres effectifs ou paramètres réels qui correspondent respectivement aux paramètres formels. On dit alors qu'il y a un passage de paramètres.

Si la fonction a une valeur de retour, son appel peut figurer dans une instruction d'affectation ou d'écriture car il constitue en lui-même une expression dont la valeur est la valeur de retour de la fonction.

Si la fonction n'a pas de valeur de retour, son appel ne peut pas figurer dans une autre instruction mais constitue plutôt une instruction à part entière.

Le passage d'un paramètre à une fonction peut se faire en entrée, en sortie ou en entrée-sortie.

Avec le passage en entrée, la valeur du paramètre effectif est une donnée que la fonction reçoit au début de l'appel dans son paramètre formel.

Le passage en entrée-sortie combine ces deux premiers modes de passage.

Il faut préciser ces modes de passage lors de la définition de la fonction en mettant le préfixe E, S ou E-S avant chaque paramètre formel. Mais si on ne fait pas cette précision pour un paramètre, son passage sera par défaut en entrée.

2. Notion de récursivité

La récursivité est le fait qu'une fonction fasse appel à elle-même, c'est à dire qui contient dans ses instructions un appel à elle-même. Une telle fonction est dite fonction récursive et l'appel est dit appel récursif.

La récursivité permet de traduire facilement des solutions de problèmes qui ont une certaine forme récurrente en algorithmes récursifs. Par exemple, en mathématique, le factoriel d'un entier n peut être défini ainsi ;:

`0 ! = 1 et si n > 0, n! = n × (n − 1) !.`

Cette définition a une forme récursive car, pour trouver le factoriel d'un entier, elle fait appel à la notion de factoriel. Elle peut être traduite en la fonction algorithmique suivante :

Fact(n :entier) :entier

Début

Si (n = = 0) Alors

Retourner 1

Sinon

Retourner n × Fact(n – 1)

FinSi

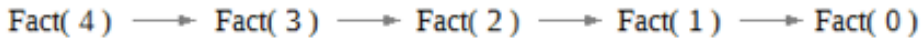
FinFonction

L'appel de cette fonction récursive Fact dans un algorithme se fait de la même manière que celui de la version non récursive.

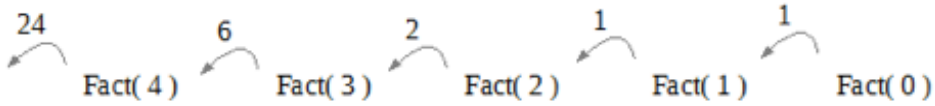
3. **Fonctionnement de la récursivité**

Pour bien comprendre ce fonctionnement, nous reprenons l'exemple précédent pour le calcul de Fact(4) dans un algorithme.

L'appel Fact(4) entraîne la suite d'appels :



et la suite de retours, qui donne finalement 24 :



En effet, pour calculer Fact(4), il faut d'abord calculer Fact(3), puis multiplier le résultat par 4, puis retourner ce résultat. Mais, pour calculer Fact(3), il faut d'abord calculer Fact(2), puis multiplier le résultat par 3, puis retourner ce résultat, et ainsi de suite.

Ainsi, pour que la récursivité fonctionne, il faut avoir des cas de base dont la solution est connue; ces cas de base constituent les cas d'arrêt de la récursivité. Il faut aussi que la suite des appels récursifs conduise à un cas de base.

Conclusion

Cette unité est une introduction l'apprenant aux algorithmes. La notion de fonction a d'abord été expliquée comme étant un regroupement d'instructions pouvant être appelé plusieurs fois dans un algorithme.

Par la suite, la notion de récursité a été expliquée par le fait qu'une fonction fasse appel à elle-même lors de son exécution. Son principe de fonctionnement a été expliqué de manière détaillée pour permettre à l'apprenant de trouver facilement des algorithmes récursifs pour certainsproblèmes de cette nature.

Évaluation

1. Lister les trois caractéristiques d'un algorithme récursif
2. Ecrire un algorithme récursif qui fait la somme de deux entiers

Résumé de l'unité

Cette unité est une introduction l'apprenant aux algorithmes. La notion de fonction a d'abord été expliquée comme étant un regroupement d'instructions pouvant être appelé plusieurs fois dans un algorithme.

Par la suite, la notion de récursité a été expliquée par le fait qu'une fonction fasse appel à elle-même lors de son exécution. Son principe de fonctionnement a été expliqué de manière détaillée pour permettre à l'apprenant de trouver facilement des algorithmes récursifs pour certains problèmes de cette nature.

Évaluation de l'unité

Vérifiez votre compréhension!

Exercice

Directives

Répondre aux questions suivantes:

1. Qu'est ce qu'un algorithme récursif?
2. Expliquer brièvement le fonctionnement d'un algorithme récursif

Système de notation

Les points seront répartis comme suit:

Question	Points
1	1
2	2 points pour chaque ligne d'explication; 8 points au maximum
Total	10

Évaluation

1. Un algorithme récursif est une méthode de simplification qui divise un problème en sous-problèmes de même nature.
2. Le résultat d'un appel récursif est une entrée pour un autre appel récursif. Ce qui implique une itération sous-jacente. Ces appels récursifs se font avec des paramètres de plus en plus petis pour conduire à un cas de base.

Lectures et autres ressources

Les lectures et autres ressorces de cette unité se trouvent au niveau des lectures et autres ressources du cours.

Unité 3. Les structures de données de base et les types abstraits de données

Introduction à l'unité

Cette unité présente les structures de données de base qu'on peut utiliser pour organiser les données dans un algorithme.Elle présente également les types abstraits qu'on peut définir à partir de ces strucures de données en spécifiant des propriétés et des opérations qui leur sont applicables.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Identifier les structures de données utilisées pour organiser les données.
- Expliquer les caractéristiques de ces structures de données.
- Décrire les types abstraits qu'on peut définir à partir de ces structures de données.

TERMES CLÉS

Structures de données: une manière d'organiser un ensemble de données dans un algorithme.

Type abstrait de données: un type défini en spécifiant un ensemble de propriétés et d'opérations sur une structure de données.

Activités d'apprentissage

Activité 3.1 - Lecture sur les structures de données

Introduction

Cette section présente les structures de données de base que sont les tableaux, les enregistrements, les listes chaînées et les arbres après avoir donné une définition générale de cette notion.

Détails de l'activité

1. Définition

Une structure de données est une manière d'organiser des données utilisées dans un algorithme. Elle peut être caractérisée par un type qui est prédéfini ou qui doit être défini par le concepteur de l'algorithme. Il existe plusieurs structures de données dont: les tableaux, les enregistrements, les listes chaînées et les arbres.

2. Les tableaux

Un tableau est une structure de données qui permet de regrouper des informations de même type appelées éléments. Chaque élément du tableau est représenté sous la forme d'une case contenant sa valeur et portant numéro appelé indice s'il s'agit d'un tableau à une dimension et deux indices s'il s'agit d'un tableau à deux dimensions.

Voici un exemple de tableau d'entiers à une dimension:

100	85	123	130	65	98
0	1	2	3	4	5

Voici un exemple de tableau de réels à deux dimensions:

	0	1
0	0.5	8.1
1	6;3	- 5.7
2	0.9	4.3

Un tableau est une structure de données dont le type est prédéfini. Il suffit de déclarer une variable de ce type pour pouvoir l'utiliser dans un algorithme.

Pour déclarer un tableau à une dimension, on peut utliser la syntaxe suivante:

id_Tableau : tableau [dimension] de type

où

id_Tableau est l'identificateur du tableau

dimension est le nombre d'éléments (la taille) du tableau

type est le type des éléments

Exemple

tab : tableau[6] d'entier

Pour un tableau à une dimension les indices varient de 0 à dimension – 1.

Un élément d'indice i du tableau est désigné par id_Tableau[i]. Par exemple,

tab[0] est le premier élément du tableau tab

tab[5] est le dernier élément du tableau tab

Un tableau à une dimension est aussi appelé vecteur.

Pour déclarer un tableau à deux dimensions, on utilise la syntaxe suivante:

id_Tableau : tableau[dimension1][dimension2] de type

Exemple

m : tableau[3][2] de réel

On dit que m est un tableau de 3 lignes et 2 colonnes

Pour chaque dimension du tableau les indices varient de 0 à dimension – 1.

id_Tableau[i][j] désigne l'élément à la ligne i et à la colonne j. Par exemple,

ml0][0] désigne l'élément situé à la ligne 0 et à la colonne 0

m[2][1] désigne l'élément situé à la ligne 2 et à la colonne 1

Un tableau à deux dimensions est aussi appelé matrice.

Un élément d'un tableau peut être utilisé de la même manière que n'importe quelle variable de même type. Autrement dit, il peut faire l'objet d'une affectation, il peut figurer dans une instruction d'écriture ou de lecture ou dans une expression.

Pour parcourir un tableau à une dimension et effectuer un traitement (remplissage, affichage, etc.), on peut utiliser une boucle.

Pour parcourir un tableau à deux dimensions, on peut utiliser deux boucles imbriquées.

3. Les enregistrements

Un enregistrement permet de regrouper des éléments qui peuvent être de types différents. Chaque élément est appelé champ ou membre et est désigné par un identificateur.

Un enregistrement est une structure de données dont le type doit être défini par le concepteur de l'algorithme. Ensuite, il pourra déclarer une variable de ce type pour pouvoir l'utiliser dans un algorithme.

Pour définir un type enregistrement, on peut utliser la syntaxe suivante:

NomEnregistrement = Enregistrement

champ1 : type1

champ2 : type2

...

champN : typeN

FinEnregistrement

où

NomEnregistrement est l'identificateur du type

type1,…, typeN sont les types des champs champ1,…, champN

Exemples

Définition d'un type pour les dates :

Type

date = Enregistrement

jour : entier

mois : chaîne

année : entier

FinEnregistrement

Après avoir défini un type enregistrement, on peut l'utiliser pour déclarer des variables. Par exemple:

d : date

On peut accéder aux champs d'un enregistrement avec l'opérateur point. Par exemple, le champ jour de l'enregistrement d est désigné par l'expression d.jour

Le champ d'un enregistrement peut être utilisé de la même manière que n'importe quelle variable de même type. Autrement dit, il peut faire l'objet d'une affectation, il peut figurer dans une instruction d'écriture ou de lecture ou dans une expression.

Les listes chaînées

Présentation

Une liste chaînée est une structure de donnée qui permettent de relier plusieurs éléments. Elle est qualifiée de structure de données dynamique car, contrairement à un tableau, ses éléments sont ajoutés au fur et à mesure par allocation dynamique. Il est donc nécessaire de garder au niveau de chaque élément un lien vers l'élément suivant dans la liste. Pour cela on utilise des pointeurs ou des références

Pointeur et allocation dynamique

Quand l'ordinateur exécute un programme correspondant à un algorithme, une variable ou une constante occupe un espace d'un certain nombre d'octets en mémoire centrale pour

contenir sa valeur. Le numéro du premier octet de cet espace est l'adresse de la variable ou de la constante.Un pointeur permet de garder une information dont la valeur est l'adresse d'un espace mémoire. On dit que le pointeur pointe sur l'espace mémoire dont il contient l'adresse. Si cet espace est occupé par une variable, on dit aussi que le pointeur pointe sur la variable. Dans un algorithme, on peut utiliser un pointeur en le déclarant comme une variable selon la syntaxe suivante :

id_pointeur : pointeur sur type

où id_pointeur est l'identificateur du pointeur qui pourra garder des adresses d'espaces mémoire contenant des données du type spécifié.

Exemple

variable

n: entier

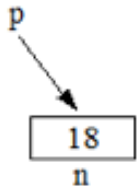
p: pointeur sur entier

n = 18

p = adresse de n

Dans cet exemple, le pointeur p contient l'adresse de la variable n.

Pour montrer que p pointe sur n, on utilise la représentation graphique suivante:



Il est possible d'accéder à un espace mémoire en passant par un pointeur qui garde son adresse. Pour cela on utilise l'opérateur *. Ainsi, avec l'exemple précédent, *p désigne la variable n.Un pointeur qui ne contient aucune adresse a pour valeur NULL. Dans ce cas, il ne pointe nulle part.Au lieu d'affecter à un pointeur l'adresse d'une variable, on peut lui affecter l'adresse d'un espace mémoire réservée de manière dynamique durant l'exécution.

En pseudo-code, l'allocation dynamique de mémoire peut se faire selon la syntaxe suivante :

id_pointeur = allouer(nb, type)

où

id_pointeur est l'identificateur du pointeur

nb est le de blocs contigus à allouer en mémoire

type est le type de données à stocker dans chaque bloc.

La libération de l'espace mémoire allouée de manière dynamique peut se faire selon la syntaxe suivante :

désallouer(id_pointeur)

Définition d'un type liste chaînée

Un élément (ou maillon) d'une liste chaînée peut être représentée par un enregistrement qui contient un champ pour la valeur de l'élément et un champ qui est un pointeur contenant l'adresse de l'élément suivant.

Le dernier élément est appelé queue de la liste. Son champ pointeur vers l'élément suivant vaut NULL.

Un pointeur sur le premier élément (tête de la liste) permet de représenter la liste chaînée.

On peut donc définir un type pour une liste chaînée d'entiers de la manière suivante:

Type

maillon = Enregistrement

valeur : entier

suivant: Pointeur sur maillon

FinEnregistrement

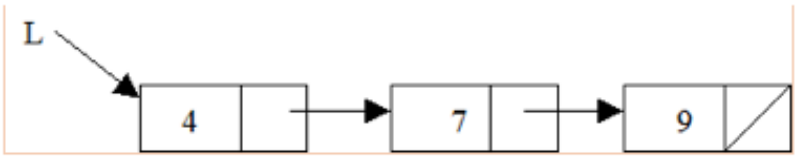
Liste = Pointeur sur maillon

Pour utiliser une liste chaînée, il faut déclarer une variable de type Liste ou bien de type Pointeur sur maillon. Par exemple :

L : Liste

Cette variable représente la liste et sert à contenir l'adresse du premier maillon. Elle vaut NULL pour une liste vide.

La figure suivante montre un exemple de liste chaînée d'entiers



Dans cet exemple, L pointe sur le premier maillon de la liste. Donc :

*L désigne ce maillon

(*L).valeur, ou L→valeur, désigne le champ valeur du maillon, ce qui correspond à la valeur 4

(*L).suivant, ou L→suivant, désigne le champ suivant du maillon, ce qui correspond au pointeur sur le deuxième maillon.

Les arbres

Un arbre est une structure de données qui permet de regrouper dans un ordre hiérarchique des informations de même type appelées d'éléments.

Un arbre peut être vide ou bien constitué d'un ou de plusieurs nœuds qui contiennent chacun une valeur qui est la valeur d'un élément.

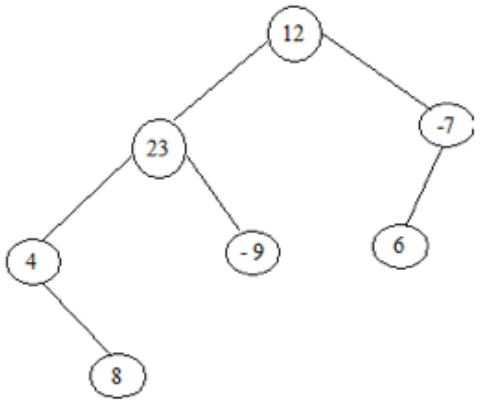
Chaque nœud de l'arbre est relié à un ou plusieurs autres nœuds placés en dessous de lui et qui sont appelés ses fils. Les feuilles sont des nœuds qui n'ont pas de fils.

Au sommet de l'arbre se trouve un nœud particulier appelé la racine.

Un arbre général (ou arbre n-aire) est un arbre dont les nœuds peuvent avoir plusieurs fils.

Un arbre binaire est un arbre dont les nœuds ont au plus deux fils. On utilise les termes de fils gauche et de fils droit d'un nœud.

La figure suivante représente un arbre binaire contenant des entiers:



Le nœud contenant la valeur 12 est la racine de l'arbre.

Le nœud contenant la valeur 23 a pour fils gauche le nœud contenant la valeur 4 et pour fils droit le nœud contenant la valeur – 9.

Le nœud contenant la valeur 4 n'a pas de fils gauche. Son fils droit est le nœud contenant la valeur 8 qui est une feuille.

On peut remarquer qu'un arbre a une forme récursive car chaque nœud autre que la racine peut être considéré comme la racine d'un sous-arbre.

Un nœud d'un arbre binaire peut être représenté par un enregistrement qui contient trois champs:

- un champ pour la valeur du nœud.

- deux autres champs qui sont respectivement un pointeur vers le nœud fils gauche et un pointeur vers le nœud fils droit.

On peut donc définir un type pour caractériser un arbre binaire d'entiers:

Type

Arbre = Pointeur sur nœud

nœud = Enregistrement

valeur : entier

filsGauche : Pointeur sur nœud

filsDroit : Pointeur sur nœud

FinEnregistrement

On peut ensuite créer un arbre en déclarant une variable de ce type:.

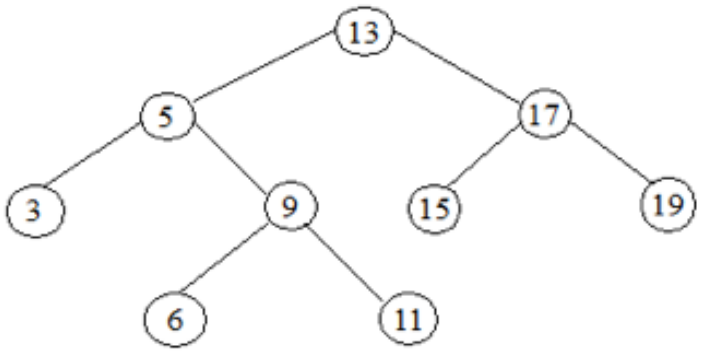
A : Arbre

Cette variable est un pointeur qui représente l'arbre et sert à contenir l'adresse du nœud racine. Donc :

*A désigne ce noeud

(*A).filsGauche, ou A filsGauche, désigne le champ filsGauche du nœud qui est un pointeur sur le nœud fils gauche

*A).filsDroit, ou A filsDroit, désigne le champ filsDroit du nœud qui est un pointeur sur le nœud fils gaucheUn arbre binaire de recherche est un arbre binaire vérifiant la propriété suivante : pour tout nœud ayant une valeur v, tous ses descendants à gauche ont des valeurs inférieures ou égales à v et tous ses descendants à droite ont des valeurs strictement supérieures à v. La figure suivante montre un exemple d'arbre binaire de recherche.



Conclusion

Après la définition de la notion de structure de données, des exemples de base que sont les tableaux, les enregistrements, les listes chaînées et les arbres ont été présentés en donnant leurs caractéristiques et la manière de les utiliser.

Évaluation

1. Donnez deux exemples d'utilisation de structures de données dans un ordinateur.
2. Qu'est ce qu'une structure de données linéaire? Donnez des exemples.

Activité 3.2 - Lecture sur les types de données abstraits

Introduction

Dans un algorithme, on peut définir un type abstrait de données par la spécification de propriétés et d'opérations applicables à une structure de donnée. Cette partie est une présentation dles types de données abstraits que sont les piles, les files et les tables de hachage.

Détails de l'activité

1. **Les piles**

Une pile est un type abstrait basé sur une structure de données qui peut être un tableau ou une liste chaînée. L'accès à la structure de données pour une opération d'ajout ou de retrait d'un élément se fait par un seul côté appelé le sommet de la pile.

Ajouter un élément se dit empiler ou push

Retirer un élément se dit dépiler ou pop

Une pile suit la règle LIFO (Last In, First Out): le dernier élément ajouté est le premier à être retiré.

2. **Les files**

Une file est un type abstrait basé sur un tableau ou une liste chaînée.L'ajout d'un élément se fait à une extrémité appelée queue et le retrait d'un élément se fait à une autre extrémité appelée tête.

Ajouter un élément à la file se dit emfiler

Retirer un élément de la file se dit défiler

Une file suit la règle FIFO (First In, First Out): le premier élément ajouté sera le premier à être retiré.

3. **Les tables de hachage**

Une table de hachage est un type abstrait basé sur un tableau de dimension m qui permet de gérer un ensemble d'éléments, chacun élément étant identifié par une clé qui est unique.

La place dans le tableau d'un élément est déterminée à partir d'un calcul effectué sur la valeur de la clé à l'aide d'une fonction de hachage h. Cette fonction associe à une clé k une valeur de hachage h(k) appartenant à l'intervalle [0,m-1]

Il peut arriver que pour deux clés différentes, la fonction de hachage donne une même valeur de hachage. On dit alors qu'il y a collision.

La stratégie de résolution des collisions dépend du type de hachage utilisé.

Dans le cas du hachage ouvert, appelé aussi hachage indirect, on utilise un tableau de listes chaînées, chaque liste contenant des éléments ayant la même valeur de hachage.

Dans le cas du hachage fermé, appelé aussi hachage direct, on cherche à résoudre une collision par un nouveau calcul de la position. Pour cela on utilise une fonction d'essais successifs qui donne à chaque essai une nouvelle place dans le tableau.

Conclusion

Dans cette activité, Après la définition de la notion de type abstrait de données, les piles, les files et les tables de hachage ont été présentés en donnant leurs spécificités. Ce sont de types abstraits de données qui ont beauoup d'applications en informatique.

Évaluation

Qu'est ce qu'un type abstrait de données?

Quelle est la différence entre une pile et une file?

Quelles sont les opérations qu'on peiut appliquer sur une pile?

Résumé de l'unité

Dans cette unité, après avoir défini une structre de données comme une manière d'organiser des données d'un algorithme, des exemples de base que sont les tableaux, les enregistrements, les listes chaînées et les arbres ont été présentés. En se basant sur une structure de données 'on peut définir type abstrait de données par la spéfication d'un ensemble de propriétés et d'opérations. Les types abstraits que sont les piles, les files et les tables de hachage ont également été présentés. Le choix d'une structure de données a une influence sur l'efficacité d'un algorithme.

Évaluation de l'unité

Vérifiez votre compréhension!

Exercice de questions-réponses

Directives

Donnez des réponses aux questions suivantes:

1. Quelle est la différence entre une structure de données et un type abstrait de données?
2. Quelle est la différence entre un tableau et une liste chaînée?
3. Qu'est ce qu'un arbre?

Système de notation

Les points seront répartis comme suit:

Question	Points
1	4
2	3
3	3
Total	10

Évaluation

1. Une structure de données est une forme d'organidsation des données d'un algorithme tandis qu'un type abstrait de données est un type caractérisant une structure de données par la spécificaftion de ses propriétes et des opérations qu'on peut lui appliquer.
2. Un tableau est une structure de données statique alors qu'une liste chaînée est une structure de données dynamiques.
3. Un arbre est une structure de données dynamique organisant des informations de manière hiérarchique.

Lectures et autres ressources

Les lectures et autres ressorces de cette unité se trouvent au niveau des lectures et autres ressources du cours.

Unité 4. Algorithmes de recherche et de tri

Introduction à l'unité

]Parfois, on peut éprouver le besoin de rechercher une information parmi d'autres ou de trier ces informations en les rangeant dans un certain ordre. Par exemple, si ces informations sont de type entier, on peut être amené à les ranger dans l'ordre croissant ou dans l'ordre décroissant. Pour cela, il existe des algorithmes de recherche ou de tri qui peuvent être appliqués aux structures de données contenant ces informations. Dans ce cours, nous allons présenter quelques uns de ces algorithmes en donnant leurs principes de fonctionnement.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de:

- Décrire le processus de recherche et de tri.
- Concevoir des algorithmes pour la recherche d'information.
- Concevoir des algorithmes pour le tri d'un ensemble d'informations.

TERMES CLÉS

Recherche: le fait de chercher d'un élément particulier dans une ensemble

Tri: le fait d'ordonner un ensemble d'éléments

Activités d'apprentissage

Activité 4.1 - Lecture sur les algorithmes de recherche et de tri

Introduction

Cette activité concerne l'étude des algorithmes de tri ou de recherche d'un élément dans un ensemble. Le résultat d'une recherche est la présence ou l'absence de l'élément recherché. Rechercher un élément revient à le comparer un à un avec les éléments d'un ensemble.Cette recherche peut se faire avec une approche séquentielle ou une approche dichotomique Quand aux algorithmes de tri, il en existe plusieurs exemples dont le tri par par sélection et le tri rapide.

Détails de l'activité

1. La recherche séquentielle

Avec la recherche séquentielle, les éléments sont comparés de manière séquentielle en commençant par le premier et en terminant éventuellemnt par le dernier si l'élément rechercé est absent ou se trouve à la fin de l'ensemble.

Ce type de recherche fonctionne pour un ensemble ordonné ou pas.

2. La recherche dichotomique

La recherche dichotomique fonctionne seulemnt avec un ensemble ordonné car elle consiste à diviser récursive ce dernier en deux parties pour trouver l'élément soit au milieu de l'ensemble ou dans l'une de ses parties selon le principe suivant:

- Si l'élément cherché est trouvé si sa valeur est égale à la valeur de l'élément au milieu
- Sinon si sa valeur est inférieur à la valeur de l'élement au milieu, alors la recherche se poursuit avec le sous-ensemble avant le mileu.
- Sinon la recherche se poursuit avec le sous-ensemble situé après le mileu.

3. Le tri par sélection

Le principe du tri par sélection consiste à déterminer successivement l'élément devant se retrouver en première position, en deuxième position, etc., c'est à dire le plus petit des éléments restants, et ainsi de suite.

Pour cela, on parcourt le tableau de gauche à droite et , à chaque position i, on place le plus petit élément qui se trouve dans le sous tableau droit (entre les positions i et N-1)

4. Le tri rapide

Le principe du tri rapide est basé sur le modèle "diviser pour régner". Pour trier un tableau tab d'indice de début deb et d'indice de fin fin, le principe est le suivant :

- Diviser: On choisit arbitrairement un élément du tableau que l'on appellera pivot. On partitionne le tableau en deux sous-tableaux en le réarrangeant de telle façon que tous les éléments inférieurs ou égaux au pivot soient placés avant lui, et que tous les éléments strictement supérieurs au pivot soient placés après lui.

- Régner: Le pivot étant à sa place définitive, il reste à trier (de la même façon) les deux sous-tableaux placés avant et après lui.

- Combiner: Les sous-tableaux étant triés sur place, on peut alors affirmer que le tableau est trié en totalité.

Conclusion

Cette activité a présenté des algorithmes de recherche et de tri en expliquant leurs principes de fonctionnement.La recherche séquentielle et la recheche dichotomique ont été expliquées de même que les algrorithmes de tri par sélection et le tri rapide.

Évaluation

1. Donnez une brève description de l'algorithme de recherche dichotomique.
2. Quelle est la différence entre la recherche séquentielle et la recherche dichotomique?
3. Donnez une brève description de l'algorithme de tri par sélection.

Question	Points
1.	6
2.	6
3.	6
Total	18

Évaluation

1. Donnez une brève description de la recherche dichotomique.

La recherche dichotomique consiste à diviser récursive ce dernier en deux parties pour trouver l'élément soit au milieu de l'ensemble ou dans l'une de ses parties

2. Quelle est la différence entre la recherche séquentielle et la recherche dichotomique?

La recherche dichotomique ne fonctionne qu'avec un ensemble ordonné alors que la recherche séquentielle fonctionne aussi bien avec un ensemble ordonné qu'un ensemble non ordonné.

3. Donnez une brève description du tri par sélection.

Le principe du tri par sélection consiste à déterminer successivement l'élément devant se retrouver en première position, en deuxième position, etc., c'est à dire le plus petit des éléments restants, et ainsi de suite.

Lectures et autres ressources

Les lectures et autres ressorces de cette unité se trouvent au niveau des lectures et autres ressources du cours.

Évaluation du cours

Résumé de l'unité

Dans cette unité, des algorithmes de recherche et de tri ont été présenté en expliquant leurs principes de fonctionnement.La recherche séquentielle fonctionne avec avec un ensemble ordonné ou pas ordonné alors que la recheche dichotomique ne fonctionne qu'avec un ensemble ordonné. Les algrorithmes de tri présentés sont le tri par sélection et le tri rapide.

Évaluation de l'unité

Vérifiez votre compréhension!

Exercice de questions-réponses

Directives

1. Donnez une brève description de l'algorithme de recherche dichotomique.
2. Quelle est la différence entre la recherche séquentielle et la recherche dichotomique?
3. Donnez une brève description de l'algorithme de tri par sélection.

Système de notation

Les points seront répartis comme suit:

Examen final

Directives

Faire les exercices suivants:

1. Donner les valeurs de a, b, c à la fin de l'exécution des instructions suivantes:

```
a = 5
b = a – 2
c = 3
d = – 2
c = a – c
a = a + d
d = c + b
```

2. Ecrire un algorithme appelé Puissance qui permet de calculer un nombre à une puissance positive donnée en utilisant une boucle. Le nombre x et la puissance n sont donnés par l'utilisateur.

Système de notation

Les points seront répartis comme suit:

Exercice	Points
1	4
2	6
Total	10

Évaluation

1. Donner les valeurs de a, b, c à la fin de l'exécution des instructions suivantes:

   a = 5

   b = a – 2

   c = 3

   d = – 2

   c = a – c

   a = a + d

   d = c + b

   A la fin: a = 3, b = 3 et c = 5

2. Ecrire un algorithme appelé Puissance qui permet de calculer un nombre à une puissance positive en utilisant une boucle. Le nombre x et la puissance n sont donnés par l'utilisateur.

Algorithme Puissance

```
Début
Variable
p , x : réel
n : entier
Ecrire( "Donnez un réel: " )
Lire( x )
Ecrire( "Donnez sa puissance: " )
```

```
Lire( n )
Si (n < 0) Alors
Ecrire( "La puissance doit être un entier positif " )
Sinon
p = 1
Pour i = 1 à n Faire
P = p × x
FinPour
Ecrire( x , " puissance ", n, " vaut ", p )
FinSi
Fin
```

Références du cours

Algorithmes en C : Cours et exercices, Robert Sedgewick, Dunod, 2005

Introduction à l'algorithmique, Cormen, Leiserson, Rivest et Stein, Dunod, 3e édition, 2010.

Algorithmes en Pascal et en langage C, Granjon, Dunod, 2004.

Initiation à l'algorithmique et à la programmation en C, Rémy Malgouyres, Rita Zrour et Fabien Feschet, Dunod, 3e édition, 2015

Algorithmique - Applications en C, Jean-Michel Léry, Pearson Education, 2005.

Algorithmique en C, Jean-Michel Léry, Pearson Education, 2e édition, 2010

Siège de l'Université Virtuelle Africaine

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

Bureau Régional de l'Université Virtuelle Africaine à Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org