Université Virtuelle Africaine

INFORMATIQUE APPLIQUÉE: CSI 1204

PRINCIPES DE PROGRAMMATION

Pélagie Houngue

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

Pélagie Houngue

Pair Réviseur

Jean Marie Dembele

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 matières

Aperçu du cours

Bienvenue aux Principes de la programmation.

Vous avez entendu parler de jeux informatiques, n'est-ce pas! Vous aviez probablement pu avoir la chance de jouer avec un ami ou avec l'ordinateur. Si ce n'est pas le cas, essayez maintenant (cela prend cinq minutes); cet ordinateur que vous utilisez, a un certain nombre de jeux, y compris et sans s'y limiter, Solitaire ou InkBall. Hou La La! Comment est-ce que cela marche? Comment juge-t-il ses mouvements? Il vous déjoue ! Que pensons-nous maintenant --- c'est un casse tête; il doit y avoir une certaine logique derrière cela! Oui, c'est cela, et ce «puzzle» est l'œuvre de la programmation. Par conséquent, ceci est un module très intéressant puisque nous introduisons ce qu'est la programmation et nous nous familiarisons avec des termes fréquemment utilisés dans le monde de la programmation. Ceci est un cours de programmation de débutant conçu pour enseigner aux élèves, les bases de la programmation. L'objectif principal du cours est d'apprendre comment résoudre efficacement les problèmes de programmation et de fournir des fondements pour des connaissances de base quel que soit le langage de programmation. Il présente les éléments constitutifs fondamentaux de la programmation tels que les variables, les opérateurs, les structures de contrôle, des tableaux, des sous-programmes, pointeurs et la gestion des fichiers. L'étudiant devra apprendre à appliquer les techniques de résolution de problèmes dans la programmation à travers la création des organigrammes et des pseudo-codes. Un langage de programmation de haut niveau (C) sera utilisé pour écrire de petits programmes afin de renforcer les notions apprises lors de la conception.

Prérequis

Ce cours n'a pas de prérequis. Cependant, les connaissances de bases en informatique constituent un avantage pour l'apprenant. C'est parce que, dans la programmation, certains éléments de l'ordinateur sont fréquemment mentionnés ou cités, par exemple la mémoire de l'ordinateur. Nous avons inclus des références, des outils, ainsi que des livres qui peuvent guider l'apprenant à travers les éléments de base d'un ordinateur.

Matériaux

Les matériaux nécessaires pour compléter ce cours comprennent:

- Un CD Rom
- Ordinateurs
- Une connexion Internet
- Logiciel de simulation.

Objectifs du cours

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

- Expliquer ce qu'est la programmation et comment elle est utilisée dans la résolution de problèmes
- Analyser les problèmes et les décomposer en unités programmables
- Concevoir des solutions aux problèmes en utilisant une méthodologie standard
- Coder la conception en utilisant le langage de programmation C.

Unités

Unité 0: Fondements de l'Informatique

Il s'agit d'une unité de pré-évaluation. Elle rappelle et vous familiarise avec certains concepts dont vous avez besoin et qui sont supposés être utilisés dans le module. Cette unité va évaluer vos compétences informatiques de base, y compris les composants matériels, logiciels et la représentation de données.

Unité 1: Introduction à la programmation

Cette unité sert de point d'entrée à la programmation. Elle comprend l'histoire de la programmation, les langages de programmation incluant les générations des langages de programmation. L'unité donne également un aperçu sur paradigmes de programmation.

Unité 2 : Résolution de problèmes basée sur l'informatique

L'unité traite des techniques de résolution de problèmes à un bas niveau, y compris l'identification, l'analyse, la conception de la solution (conception descendante de programme ou Top down et conception ascendante de programme ou bottom up). Elle explique également comment traduire des solutions dans des organigrammes et / ou pseudo-codes.

Unité 3: Programmation en langage C

Cette unité présente les bases de la programmation telles que les variables, des opérateurs, des structures de contrôle, des tableaux et des sous-programmes. Des programmes simples sont écrits en langage C pour renforcer toutes les notions apprises dans cette unité. Une activité de laboratoire est proposée dans cette unité.

Unité 4: Tests de programme, débogage et documentation

Cette unité discute des différents types d'erreurs qui apparaissent au cours du développement d'un programme et comment les déboguer. Elle aborde également les bases de la documentation en matière de développement de logiciels.

Évaluation

Les évaluations formatives (vérification de progrès) sont inclues 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	Examen à mi-parcours	10%
2	Missions	10%
3	Pratique	10%
4	Examen final	70%

Plan

Unité	Sujets et Activités	Durée estimée
Unité 0 : Evaluation diagnostique	Evaluation	2h
Unité 1 : Introduction à la programmation	Activité 1.1 - Histoire de la programmation Activité 1.2 - Les niveaux, les générations et les paradigmes des langages de programmation Activité 1.3 -Activité en équipe (Collaboration) Evaluation	5h 6h 8h 2h
Unité 2 : Résolution de problème basée sur l'informatique	Activité 2.1 - Résolution de problèmes Activité 2.2 - Conception de programme Activité 2.3 - Activité de recherche en groupe Evaluation	5h 5h 8h 2h

Unité 3 : Programmation en langage C	Activité 3.1 - Types de données, variables et opérateurs Activité 3.2 - Structures de contrôle et Fonctions Activité 3.3 – Les tableaux et les chaînes de caractères Activité 3.4- Activité de laboratoire Evaluation	10h 15h 15h 7h 2h
Unité 4.Test de programme, Débogage et Documentation	Activité 4.1 – Erreurs de programme et test Activité 4.2 – Débogage de programme Activité 4.3 – Documentation de programme/logiciel Evaluation	6h 6h 8h 2h
Examens	Examen de mi parcours et examen final	6h
	Total	**120h**

Lectures et autres ressources

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

Unité 0

Lectures et autres ressources obligatoires:

- Understanding Computers in changing society. Deborah Morley. 6th edition. Cengage Learning, 2014. 1305177045, 9781305177048
- Introduction to computers, Rajmohan J., Gyan publishing House, 2006 pp 16-28
- Computers: Tools for an Information Age, Capron, H. L. & Johnson, J. A. 8th Ed. 2004, Prentice Hall.

Lectures et autres ressources optionnelles:

- Introduction to computers. Gary B. Shelly, Steven H. Freund, Misty E. Vermaat, 8th Ed. 2010. Cengage Learning.

Unité 1

Lectures et autres ressources obligatoires:

- A brief history of computing.Gerard O'Regan, 2nd edition, 2012. Springer London.
- Programming Paradigms and Methodology.Seema Kedar,3rd revised edition, 2007. Technical Publication Pune.

Lectures et autres ressources optionnelles:

- Concepts in programming Languages. John C. Mitchell. Cambridge University Press, 2003. Chapter 1.
- http://archive.oreilly.com/pub/a/oreilly//news/languageposter_0504.html
- http://www.softpanorama.org/History/lang_history.shtml#General
- Principles of programming languages. R. D. Tennent. Prentice/Hall International 1981.

Unité 2

Lectures et autres ressources obligatoires:

- Fundamentals of Programming: with Object Oriented Programming, Gary Marrel, 2009, Gary Marrer.
- Programming Logic and design, Comprehensive, Joyce Farrel, 2012, 7th edition. Cengage Learning.
- Programming and problem Solving Through "C" Language. Harsha Priya, & R. Ranjeet. Firewall Media, 2006.

Lectures et autres ressources optionnelles:

- Problem solving and program design in C. Jeri R. Hanly & Elliot B. Koffman, 6, illustrated, Addison-wesly, 2009.
- Programming & Prob solving using C. ISRD, Tata McGraw-Hill Education

Unité 3

Lectures et autres ressources obligatoires:

- Fundamentals of programming languages. Dipali P. Baviskar, Technical Publication Pune.
- Programming and problem solving through "C" languages. Harsha Priya, r. Ranjeet. Firewall media, 2006.
- Programming and PROB solving using C. ISRD, Tata McGraw-Hill education.

Lectures et autres ressources optionnelles:

- Fundamentals of Programming: with Object Oriented Programming, Gary Marrel, 2009, Gary Marrer.
- Programming Logic and design, Comprehensive, Joyce Farrel, 2012, 7th edition. Cengage Learning.
- Programming: Grade in Industrial Technology Engineering. Creative Commons. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf

Unité 4

Lectures et autres ressources obligatoires:

- Fundamentals of programming languages. Dipali P. Baviskar, Technical Publication Pune.
- Programming and problem solving through "C" languages. Harsha Priya, r. Ranjeet. Firewall media, 2006.
- Programming and PROB solving using C. ISRD, Tata McGraw-Hill education.

Lectures et autres ressources optionnelles:

- Fundamentals of Programming: with Object Oriented Programming, Gary Marrel, 2009, Gary Marrer.
- Programming Logic and design, Comprehensive, Joyce Farrel, 2012, 7th edition. Cengage Learning.
- Programming: Grade in Industrial Technology Engineering. Creative Commons. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf

Unité 0. Évaluation diagnostique

Introduction à l'unité

Ce test initial évalue les aptitudes des apprenants sur les bases de l'informatique. Il sera également destiné à rafraîchir la mémoire des apprenants sur les éléments importants de l'ordinateur. Cela permettra aux apprenants de cerner rapidement les unités suivantes du cours. Cela ne devrait pas être un gros problème surtout pour les apprenants qui interagissent pour la première fois avec l'ordinateur, ce qui signifie qu'ils n'ont pas besoin de connaissances préalables dans le domaine de l'informatique car divers liens et références sont fournies (disponible dans la section lecture et autres ressources de cette unité) pour leur permettre d'apprendre les notions de base de l'informatique.

Objectifs de l'unité

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

1. Enumérer et expliquer les fonctions de base des composants matériels de l'ordinateur
2. Décrire comment un ordinateur représente des données
3. Expliquer les types de logiciels

Termes clés

Un Ordinateur: Ce sont des machines électroniques qui acceptent les données et les instructions d'un utilisateur, stockent les données et les instructions, manipulent les données selon les instructions et stockent et /ou affichent les résultats à l'utilisateur.

Calculateurs numériques: Ils représentent les données sous forme de valeurs discrètes en agissant sur elles suivant des étapes.

Calculateurs analogiques: Ils traitent les données sous la forme de tensions électriques.

Hardware: Ce sont les pièces tangibles, physiques que nous pouvons voir et toucher.

Unité Centrale de Traitement: un microprocesseur qui interprète et exécute les instructions données par le logiciel. Il contrôle les composants de l'ordinateur.

Unité arithmétique et logique: C'est un dispositif informatique qui effectue les opérations arithmétiques ou logiques.

L'unité de commande: C'est un dispositif qui coordonne et contrôle toutes les pièces du système informatique.

Mémoire de l'ordinateur: C'est le terme utilisé pour décrire les dispositifs qui permettent à l'ordinateur de conserver des informations. Des instructions et des données de programme sont stockées dans des puces de mémoire pour un accès rapide par le CPU.

Mémoire primaire: C'est un lieu de stockage temporaire pour les données, instructions et informations. La mémoire stocke le Système d'Exploitation, les programmes d'application et les données traitées par les programmes d'application.

Adresse mémoire: C'est un indice qui identifie de manière unique un octet de mémoire ou cellule.

Mémoire secondaire: il s'agit d'un périphérique de stockage externe qui n'est pas sur la carte mère, mais qui se trouve soit à l'intérieur ou à l'extérieur de l'ordinateur et qui stocke les programmes et données informatiques de façon permanente ou semi-permanente.

Bus d'ordinateur: C'est un canal électrique qui permet à divers dispositifs intérieurs et attachés à l'unité de système de communiquer les uns avec les autres ou de transmettre des signaux / informations.

Système de numération binaire: Il s'agit d'un système de numérotation que l'ordinateur utilise pour représenter des données à l'aide de deux valeurs uniques; 0 ou 1.

Un registre: C'est un dispositif de stockage qui stocke temporairement les instructions et les données les plus fréquemment utilisées.

Un bit: il est considéré comme l'unité de base de l'information; représenté par 1 et 0 (numéros binaires).

Byte: Par exemple, huit bits sont regroupés pour représenter un caractère, une lettre alphabétique, un nombre ou un symbole de ponctuation. Il comprend 256 combinaisons différentes.

Logiciel: Aussi connu sous le nom de programme, c'est une séquence d'instructions exécutées pour obtenir un résultat.

Logiciel système: Tous les programmes liés aux opérations informatiques de coordination par exemple les systèmes d'exploitation, les traducteurs de langage de programmation et programmes utilitaires.

Système d'exploitation: C'est un ensemble de programmes contenant des instructions qui coordonnent toutes les activités entre les dispositifs de matériel informatique.

Le logiciel d'application: Il s'agit de programmes qui exécutent des tâches spécifiques pour les utilisateurs, comme un programme de traitement de texte, un programme e-mail, ou un navigateur Web.

Évaluation de l'unité

Évaluation

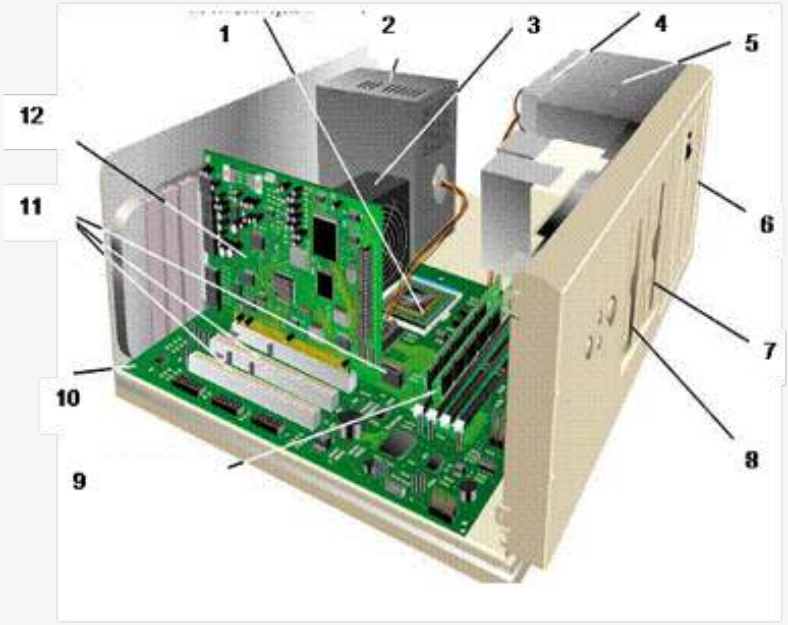
1. Qu'est-ce qu'un ordinateur? (**2 points**)
2. Il traite les données de manière discrète. Quel ordinateur est-ce? (**2 points**)

   a. Analogique b. numérique c. Hybride d. Analogiques et numériques
3. Un ordinateur accède aux informations de la mémoire principale à l'aide; (**2 points**)

   a. Un octet unique b. Un nom unique c. Une adresse unique
4. Lequel / s est une périphérique d'entrée (3 **points**)

   a. Souris b. Microphone c. Traceur d. Lecteurs de disque

5. Nommer toutes les pièces d'un ordinateur (**12 points**)



6. La mémoire principale de l'ordinateur fait : (**3 points**)

   a. Stocker tout ou partie du programme qui est en cours d'exécution

   b. Traiter les données

   c. Stocker les programmes du Système d'Exploitation qui gèrent les opérations de l'ordinateur

   d. Garder les données que le programme utilise

   e. Stocker les informations permanente

7. Un traducteur de langage de programmation est; (2 points)

   a. Un système d'exploitation b. Un logiciel utilitaire c. Un logiciel système

   d. aucun

Réponse de l'évaluation

1. Ce sont des machines électroniques qui acceptent les données et les instructions d'un utilisateur, stockent les données et les instructions, manipulent les données selon les instructions et stockent et /ou affiche les résultats à l'utilisateur.
2. a
3. c
4. a, b, d
5. 1. CPU 2. Alimentation 3. Ventilateur 4. Disque dur 5. baies de stockage 6. lecteur de CD DVD 7. lecteur de disquette 8. lecteur Zip 9. Mémoire vive (RAM) 10. Carte mère 11. Fente d'extension 12. Carte d'extension
6. a, c, d
7. c

Lectures et autres ressources

1. Understanding Computers in changing society. Deborah Morley. 6th edition. Cengage Learning, 2014. 1305177045, 9781305177048
2. Introduction to computers, Rajmohan J., Gyan publishing House, 2006 pp 16-28
3. Computers: Tools for an Information Age, Capron, H. L. & Johnson, J. A. 8th Ed. 2004, Prentice Hall.
4. Introduction to computers. Gary B. Shelly, Steven H. Freund, Misty E. Vermaat, 8th Ed. 2010. Cengage Learning.

Unité 1. Introduction à la programmation

Introduction à l'unité

Les ordinateurs sont partout. Que ce soit les grands magasins, les hôtels, les centrales atomiques ou à la défense. Les ordinateurs aident les humains dans l'administration efficace d'une tâche. La croissance des ordinateurs est exponentielle et on n'a besoin de bons programmeurs. Par "bons programmeurs" nous n'entendons pas les programmeurs qui peuvent donner des solutions, mais les programmeurs qui peuvent donner des solutions efficaces. Les bons programmeurs ne sont pas nés bons, au contraire, ils ont été formés. "Les compétences d'un programmeur se révèlent bien plus avec l'expérience et la pratique", Harsha Priya, R. Ranjeet. Dans cette unité, vous allez apprendre les concepts de base de la programmation et par la même occasion étudierez l'historique des langages de programmation.

Objectifs de l'unité

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

1. Expliquez ce qu'est la programmation et de définir les concepts de base de la programmation
2. Décrire l'évolution des langages de programmation et leurs influences sur l'évolution de la conception des langages
3. Décrire les niveaux et les générations de langages de programmation
4. Décrire les caractéristiques d'un bon langage
5. Décrire et expliquer les caractéristiques des paradigmes des langages de programmation

Termes clés

La programmation est l'activité d'écriture d'instructions devant être exécutée par l'ordinateur. Ces instructions indiquent à l'ordinateur de faire quelque chose. Par exemple, résoudre un problème particulier, lire les notes des étudiants et calculer le rang par étudiant et de fournir une liste d'étudiants qui passent en classe supérieure et une autre liste des étudiants qui recommenceront le cours. Tout cela, ce sont des instructions (logiques). Un ordinateur n'est pas en mesure d'agir sans instructions. Aussi, la programmation est l'art d'exécuter des tâches de calcul et de convertir ces tâches sous une forme lisible par la machine (Harsha Priya, R. Ranjeet)

Le langage de programmation est une série d'instructions pour écrire des programmes.

Les paradigmes des langages de programmation définissent la manière dont les programmes sont structurés ou les styles de programmation.

Un programmeur est la personne qui écrit des instructions ou développe un programme en utilisant la programmation.

Un programme est une série d'instructions qui demande à l'ordinateur de faire quelque chose. C'est le produit final de l'activité de programmation, aussi connu sous le nom de logiciel.

La syntaxe d'un langage informatique est l'ensemble de règles qui définit les combinaisons de symboles qui sont considérés comme un document ou un fragment correctement structuré dans cette langue (http://en.wikipedia.org).

La sémantique définit le sens syntaxique légal des chaînes de caractères définies par un langage de programmation spécifique, montrant le calcul impliqué (http://en.wikipedia.org).

Activités d'apprentissage

Activité 1.1 - Histoire de la programmation

Introduction

La programmation peut être définie comme l'action d'utiliser un langage de programmation pour écrire des instructions qui peuvent être exécutées par l'ordinateur dans le but de résoudre des problèmes en appliquant correctement la syntaxe et la sémantique du langage. Des centaines de langage de programmation ont été développés et certains existent encore tandis que d'autres n'existent plus. Cela signifie que le développement des langages de programmation a parcouru un long chemin et a ainsi une longue histoire; de Plankalkül, Prolog, FORTRAN, C, Visual Basic.Net, à Rust et Swift. Certains de ces langages ont été améliorés et ont gagné plus de popularité que d'autres. Ci-dessous, nous avons un résumé de l'histoire des langages de programmation tel que présentée par Seema Kedar.

1951-1955 : Utilisation expérimentale des compilateurs d'expression.

1956-1960 : FORTRAN, COBOL, LISP, Algol 60.

1961-1965 : la notation APL, Algol 60 (révisée), SNOBOL, CPL.

1966-1970 : APL, SNOBOL 4, FORTRAN 66, BASIC, SIMULA, Algol 68, Algol-W, BCPL.

1971-1975 : Pascal, PL/1 (Standard), C, Scheme, Prolog.

1976-1980: Smalltalk, Ada, Fortran 77, ML.

Détails de l'activité

Dans cette activité, vous devez lire l'histoire de la programmation, c'est-à-dire comment la programmation a commencé et l'application au programme, leur descente vers les langages actuels, les influences sur l'évolution de la conception et le développement des langages. Les informations peuvent être obtenues à partir de "Programming Paradigms and Methodology" de Seema Kedar et "Programming and PROB using C" de ISRD et d'autres références énumérées dans la section référence de cette unité et des informations connexes et pertinentes disponibles sur Internet. Les apprenants devraient également compléter les lectures en faisant des recherches individuelles afin d'améliorer leur compréhension. Dans cette activité, vous devez:

- Lister au moins cinq problèmes qui peuvent être résolus à l'aide d'un ordinateur (juste pour vérifier la compréhension de ce qu'est la programmation)
- Décrire comment la programmation a commencé et les situations qui ont conduit au développement des programmes
- En utilisant un schéma, décrire la prolifération des langages de programmation; Nommer et décrire les utilisations et la signification des langages de programmation. Montrer et décrire tous les descendants d'un langage particulier.

Conclusion

Cette activité aidera les apprenants à apprécier l'histoire de la programmation et peut également stimuler les apprenants à comprendre comment tirer parti des situations et des environnements tels qu'ils se présentent pour proposer et concevoir de nouveaux ou d'améliorer les langages existants.

Évaluation

Vous pouvez faire cette évaluation individuellement ou en groupe avant d'essayer de faire correctement les évaluations sommatives. Cette évaluation porte sur l'activité qui vient de s'achever.

- Selon ce que vous savez sur l'histoire de l'informatique, quand a été conçu le premier langage de programmation et quelle était son utilisation?
- Quelle est l'importance d'étudier l'histoire de la programmation ?
- Qu'est ce qui a influencé le développement des langages de programmation C et Fortran ?
- Maintenant, vous êtes au courant que des centaines de langages de programmation existent. Certains sont largement utilisés dans l'industrie aujourd'hui et certains ne sont jamais du tout mentionnés ou sont utilisés par un petit groupe de programmeurs. Pourquoi pensez-vous que la plupart de ces langages ont gagné plus de popularité que d'autres?

Activité 1.2 - Les niveaux, les générations et les paradigmes des langages de programmation

Présentation

Les programmeurs utilisent le langage de programmation pour écrire des programmes et les langages de programmation ont évolué au fil du temps avec les premiers langages connus également comme langages de première génération utilisant le code machine pour programmer l'ordinateur datant de 1940 à 1950. Ce langage est qualifié de langage de bas niveau. Le développement suivant a été les langages de deuxième génération datant de la période de 1950 à 1958 qui utilisent le langage d'assemblage pour représenter des instructions en langage machine. Elles sont ensuite traduites en code machine par un assembleur. Ce langage était aussi un langage de bas niveau. Ensuite, les langages de troisième génération qui inclut le développement des langages de programmation de haut niveau tels que C, Pascal, FORTRAN et COBOL, et datant de la période 1958 à 1985. Ils étaient plus faciles à utiliser que les langages d'assemblage et le code machine et ont contribué à améliorer la qualité et la productivité. Les compilateurs et les interpréteurs sont utilisés pour traduire les instructions d'un langage de haut niveau vers le langage machine.

Il y a également une quatrième génération de langages à partir de 1985. Ces langages incluent les générateurs de rapports, qui réduisent ainsi l'effort de programmation. Les langages de programmation de cinquième génération datent à partir de 1990 et sont principalement utilisés dans le domaine de l'intelligence artificielle. Chaque génération a ses propres caractéristiques et une nouvelle génération de langage est plus améliorée qu'une ancienne. Un langage de programmation s'inscrit dans un paradigme particulier ou supporte plus d'un paradigme pour concevoir et mettre en œuvre un programme. Le paradigme définit la façon dont le programme est structuré. La figure ci-dessous illustre les niveaux de langages de programmation.

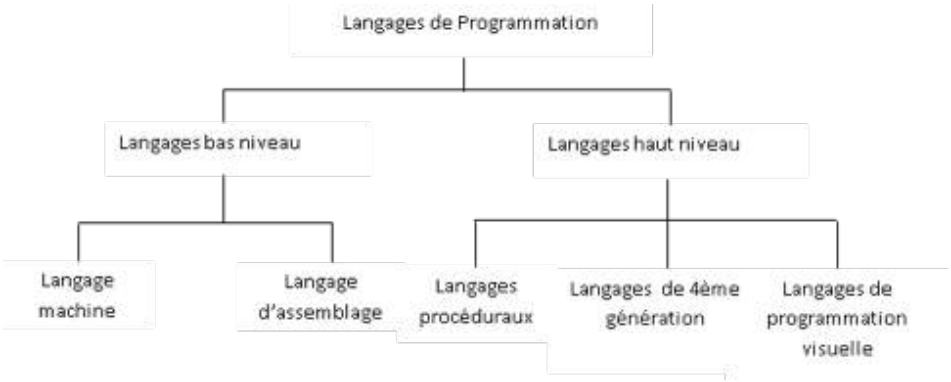


Figure 1.1 : Les niveaux de langages de programmation
(Source: ISRD)

Le schéma montre qu'il y a deux grands niveaux de langages de programmation : les langages de programmation de bas niveau et de haut niveau.

Langage machine: En utilisant ce langage, les programmes sont écrits en utilisant des 0 et des 1. Il a été utilisé au départ pour programmer un ordinateur. Il est plus rapide parce que les instructions sont directement exécutables, et il fait un usage plus efficace des ressources comme les registres et les unités de stockage. Actuellement, ce langage n'est pas très apprécié parce que les programmes ne sont pas portables, c'est-à-dire ils sont dépendants de la machine, il est plus sujet à des erreurs et difficile à déboguer, il nécessite des compétences de programmation de haut niveau et ainsi un coût élevé pour la formation.

Langage d'Assemblage: Ce langage utilise des mnémoniques ou de courtes abréviations représentant une instruction. Il est plus facile à utiliser que le langage machine simplement parce les mnémoniques sont plus proches du programmeur que de l'ordinateur (machine). L'assembleur est utilisé pour convertir les instructions en langage machine.

Langages de haut niveau: Les programmes sont écrits en anglais comme des déclarations. Ils ne sont pas directement exécutables donc les traducteurs qui comprennent les compilateurs et les interpréteurs sont utilisés pour convertir les instructions en langage machine. Les langages qui appartiennent à cette catégorie sont très appréciés car ils sont portables, faciles à apprendre et pour écrire un programme, la disponibilité des bibliothèques, la facilité de maintenance et de documentation. Les langages de haut niveau ont différents attributs ou caractéristiques qui comprennent : la portabilité, la lisibilité, le support de la concurrence, le support du mixage de langage, la fiabilité, la modularité, le support du temps réel et l'orthogonalité.

Langages de quatrième génération: Ces langages sont aussi connus comme 4GL et soulignent sur ce qui doit être accompli plutôt que la façon de l'accomplir. Comme exemple nous avons Oracle, SQL, etc. Ils sont principalement utilisés pour accéder aux bases de données. Les 4GL augmentent la productivité. Ils ont diverses caractéristiques qui les rendent attrayants. Il s'agit notamment de :

- la facilité d'utilisation
- le nombre restreint de fonctions
- la disponibilité d'options
- les options par défaut.

Les compilateurs

Un compilateur traduit un programme de haut niveau en langage machine. Le programme de haut niveau est appelé code source, qui se traduit pour produire un code lisible par la machine connu sous le nom de code objet. La plupart ou tous les langages de haut niveau utilisent un compilateur. Par exemple, le compilateur C, C++, etc.

Les Interpréteurs

Tout comme les compilateurs, les interpréteurs traduisent le programme écrit dans un langage de haut niveau en un format lisible par la machine. La différence avec les compilateurs, c'est que, les interpréteurs traduisent le code déclaration par déclaration et si une erreur est rencontrée, il s'arrête et poursuit après que l'erreur ait été corrigée. Alors que les compilateurs traduisent l'ensemble du programme, et ensuite listent les erreurs, le cas échéant. Cela signifie que les compilateurs sont plus rapides que les interpréteurs.

Influences sur l'évolution de la conception des langages

Seema Kedar a présenté six influences :

- Capacités des ordinateurs: Les ordinateurs ont grandement amélioré la façon dont ils traitent l'information. Il y a eu des améliorations majeures sur le matériel et dans le même temps au niveau logiciel (systèmes d'exploitation). Ces améliorations participent à la conception de meilleurs langages.
- Applications: Application signifie l'utilisation des ordinateurs. Ils ont commencé par faire de petits calculs, en manipulant des activités militaires, des activités scientifiques, comme la recherche, la médecine, des affaires aux jeux et l'Internet. Les exigences des nouveaux domaines d'application conduisent à l'amélioration des langages existants ou à la conception de nouveaux langages
- Méthodes de programmation : Comme les problèmes deviennent plus complexes, de nouvelles méthodes sont inventées pour aider à réduire la complexité et améliorer la productivité.

- Méthodes d'implémentation : Le développement de meilleures méthodes de mise en œuvre a affecté le choix des caractéristiques à inclure dans les nouveaux modèles de langage
- Études théoriques : La recherche dans la base conceptuelle pour la conception et la mise en œuvre des langages en utilisant des méthodes mathématiques formelles a augmenté notre compréhension des forces et faiblesses des fonctionnalités des langages qui ont influencé l'inclusion de ces fonctionnalités dans la conception de nouveau langage
- Normalisation: La nécessité d'un langage standard qui peut être facilement mis en œuvre en utilisant un système d'ordinateurs permettant aux programmes d'être traduits d'un ordinateur à un autre, a exercé une forte influence sur l'évolution de la conception des langages.

Les langages ont différentes forces. Certains langages soutiennent des applications particulières que d'autres. Qu'est-ce donc, définir un bon langage de programmation?

- Clarté, simplicité et unité
- Orthogonalité
- Naturalité des applications
- Support pour l'abstraction
- Facilité de vérification du programme
- Environnement de programmation
- Portabilité des programmes
- coût
  - coût de la formation
  - coût de l'écriture du programme
  - coût de l'exécution du programme
  - coût de la traduction
  - coût de mise en œuvre
  - coût de l'entretien

Les paradigmes des langages de programmation

Les langages de programmation sont structurés de différentes manières. Par conséquent les langages peuvent être regroupés en fonction de la façon dont ils sont structurés ou conçus, simplement connue comme paradigme. Les paradigmes comprennent;

Langages impératifs ou procéduraux: Ces langages sont basés sur des commandes et chaque exécution d'une commande modifie les valeurs dans la mémoire. L'exécution des commandes est séquentielle.

Langages fonctionnels: Ce paradigme est orienté fonction. Il ne met pas l'accent sur la séquence des événements qui conduisent à des changements de valeurs de mémoire, mais plutôt s'intéresse à ce que la déclaration peut faire.

Langages logiques ou basés sur les règles : Ils supportent la prise de décision basée sur les conditions fournies

Langages orientés objet: C'est le paradigme qui fournit des solutions qui représentent le scénario du monde réel. Il met l'accent davantage sur des données. Il utilise le concept de classes où un objet est considéré en termes de données (ce qu'il peut traiter) et ce qu'il peut faire (méthodes).

Langages concurrents: Ils appliquent le concept d'un processus. Un processus correspond à un calcul séquentiel, avec son propre thread de contrôle.

Détails de l'activité

Dans cette activité, vous devez lire sur les niveaux (d'autres auteurs appellent cela types de langages de programmation) et les générations de langages de programmation; constater les caractéristiques, les avantages et les inconvénients de chaque type/niveau, y compris les générations de langages de programmation. Il faut lire sur les caractéristiques d'un bon langage de programmation. Il faut lire également sur le paradigme des langages de programmation; les caractéristiques de chaque paradigme y compris les langages qui correspondent à chaque paradigme. Nous recommandons "Programming Paradigms and Methodology" écrit par Seema Kedar et "Programming and PROB using C" écrit par ISRD. Après les avoir lu, écrire de brefs résumés.

- Décrire les niveaux et les générations de langages de programmation et citer les langages de programmation qui appartiennent à chaque niveau et génération.
- Décrire les caractéristiques des langages de programmation qui appartiennent à un niveau et génération particulier.
- Décrire les caractéristiques d'un bon langage de programmation. Comment un bon langage de programmation se définit?
- Décrire les caractéristiques des langages de programmation qui sont regroupés au sein d'un paradigme de programmation particulier.

Activité pratique

- Manuellement ou en utilisant un logiciel, élaborer un schéma qui regroupe tous les langages de programmation identifiés dans l'activité 1.1 selon le niveau et la génération de langages de programmation correspondant.
- Manuellement ou en utilisant un logiciel, élaborer un schéma qui regroupe les langages de programmation identifiés dans l'activité 1.1 et selon la catégorie de paradigme de programmation à laquelle ils appartiennent.

Conclusion

A ce stade du cours, les apprenants doivent avoir un bon niveau de connaissances sur les caractéristiques des niveaux, des générations et des paradigmes des langages de programmation. Ils devraient être en mesure de décrire les langages de programmation qui appartiennent à chaque niveau, génération et paradigme.

Évaluation

Vous pouvez faire cette évaluation individuellement ou en groupe avant de tenter de remplir correctement les évaluations formatives. Cette évaluation est sur l'activité qui vient de s'achever.

1. Le langage machine est connu comme un langage de bas niveau et même si l'assembleur utilise des mnémoniques, il est toujours considéré comme un langage de bas niveau. Pourquoi pensez-vous cela?
2. Les langages de programmation sont classés suivant des générations. Pourquoi?
3. Décrire au moins quatre caractéristiques d'un bon langage de programmation. Le Langage de haut niveau peut être traduit en langage machine en utilisant soit un compilateur ou un interpréteur. Quels sont les avantages et les inconvénients d'un compilateur et d'un interpréteur. Lequel des deux traducteurs est plus efficace et pourquoi?
4. Décrire les cinq générations de langages de programmation.
5. Qu'est ce qui a influencé le développement des paradigmes de programmation?
6. Que pensez-vous que sera l'avenir des langages de programmation. Pensez-vous que nous aurons plus de générations et de paradigmes de langages de programmation?

Activité 1.3 -Activité en équipe (Collaboration)

Introduction

Cette activité implique une activité de groupe où les apprenants seront amenés à comprendre la pratique des langages de programmation et des paradigmes. Les apprenants sont tenus de faire une recherche à partir des développeurs individuels ou des entreprises sur les langages couramment utilisés et les paradigmes supportés par les langages.

Détails de l'activité

En visitant les développeurs individuels et / ou des entreprises :

- Donnr les langages de programmation les plus populaires dans l'industrie et les paradigmes supportés par les langages. Donnez les raisons de leur popularité.
- Dire comment les programmeurs sélectionnent le langage de programmation à utiliser dans la résolution d'un problème.

Conclusion

Le but de cette activité est d'élargir les capacités d'innovation des apprenants et à la fin l'apprenant devrait être en mesure d'apprécier les utilisations de divers langages et paradigmes et être capable de travailler en collaboration avec les autres.

Évaluation

Vous pouvez faire cette évaluation individuellement ou en groupe avant de tenter de remplir correctement les évaluations formatives. Cette évaluation est sur l'activité vient de s'achever.

Lors de votre activité de recherche, quel est le langage qui est commun et le plus préféré par les développeurs?

- Quelles sont quelques raisons de la préférence du langage mentionné (à la question 1 ci-dessus).
- Comment les programmeurs choisissent le langage à utiliser pour résoudre un problème particulier.

Résumé de l'unité

Cette unité a permis d'apprendre ce qu'est la programmation, l'histoire de la programmation, les niveaux, les générations et les paradigmes des langages de programmation. Jusqu'ici, l'apprenant comprend les concepts de base de la programmation et est prêt à passer à l'unité suivante.

Évaluation de l'unité

Système de notation

Question 1 : 5 points

Question 2 : 2 points

Question 3 : 1 point

Question 4 : 2 points

Question 5 : 1 point

Instructions

Le but de cette évaluation est de vérifier les progrès de l'apprenant en déterminant ce que l'apprenant a appris dans cette unité. Les questions couvrent tout ce qui a été présenté dans cette unité afin d'évaluer la compréhension globale. Répondez attentivement et si votre score tombe

- en dessous de 40%, refaire les lectures.
- entre 40% et 60%, refaire les lectures sur votre zone de faiblesse
- supérieur à 60%, vous avez une quantité importante de connaissances

Questions

1. La programmation peut être définie comme le fait d'utiliser un _____ pour écrire des_____ qui peuvent être exécutées par l'ordinateur dans le but de _____ en appliquant les _____ et _____ correctes du langage de programmation.
2. _____ a été la première génération de langage d'ordinateur et utilise le langage _____ pour écrire les instructions
3. Lequel des langages suivants utilise des mnémoniques pour représenter les instructions

   a. Langage de haut niveau b. langage de bas niveau c. langage assemblage
4. Un langage de haut niveau utilise _____ ou _____ pour traduire les instructions en langage machine.
5. Lesquels des langages suivants supportent le paradigme orienté objet

   A. C B. C ++ C. Pascal D. Java E. Visual Basic.Net

Réponses

1. langage de programmation, instructions, résoudre un problème, les syntaxes, la sémantique
2. Le langage machine, bas niveau
3. Assemblage,
4. des compilateurs ou des interpréteurs
5. C++, Java, Visual Basic.Net

Lectures et autres ressources

1. Seema Kedar, 2007. Programming Paradigms and Methodology . Technical Publication Pune; 3e édition révisée
2. Programming and PROB using C. ISRD, Tata McGraw-Hill Education
3. Gerard O'Regan, 2012. A brief history of computing. Springer London; 2e édition.
4. Concepts in programming Languages. John C. Mitchell. Cambridge University Press, 2003. Chapitre 1.
5. http://archive.oreilly.com/pub/a/oreilly//news/languageposter_0504.html
6. http://www.softpanorama.org/History/lang_history.shtml#General

Unité 2. Résolution de problème basée sur l'informatique

"The sooner you start coding your program, the longer it is going to take" [H.F. ledgard, programming proverb]

Introduction de l'unité

Dans l'unité précédente, nous avons défini la programmation comme le moyen d'utiliser les ordinateurs pour résoudre les problèmes avec le seul but de rendre le travail facile et augmenter la productivité. Toujours dans l'activité 1, nous étions chargés d'identifier quelques problèmes qui peuvent être résolus à l'aide d'un ordinateur. Cette unité vous aidera à découvrir si vous les avez bien fait ou non. Ainsi la programmation implique l'automatisation des systèmes, des processus, des opérations ou une activité. La solution (programme) doit satisfaire les besoins ou les exigences de l'utilisateur final. L'utilisateur final peut être un agriculteur dans le village qui a besoin des mises à jour quotidiennes sur les prix du marché, y compris des informations qui peuvent aider à améliorer la production. Par conséquent, pour une solution à réaliser, les programmeurs doivent d'abord comprendre le problème et suivre les processus de développement de logiciels qui entraînent, l'identification des problèmes, l'analyse, la conception de la solution qui peut être réalisé grâce à l'utilisation des techniques de conception descendante (top down) ou ascendante (bottom up). Les solutions peuvent également être conçues en utilisant des organigrammes (flowchart) et des algorithmes et / ou pseudocodes. Par conséquent, cette unité se focalise sur l'analyse des problèmes et les techniques de conception de solutions à un niveau très basique.

Objectifs de l'unité

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

1. Formuler et définir les problèmes.
2. Concevoir des solutions en appliquant les techniques de conception.
3. Concevoir des organigrammes (flowchart).
4. Décrire les règles de développement d'un organigramme.
5. Représenter des solutions en utilisant des algorithmes et pseudocodes.
6. Décrire les caractéristiques d'un algorithme et d'un pseudocode.
7. Décrire les techniques de programmation.

Termes clés

Algorithme: Un algorithme est une procédure consistant en un ensemble fini de règles claires (instructions) qui spécifient une séquence finie d'opérations qui fournit la solution à un problème ou à une classe spécifique de problèmes pour tout ensemble admissible de variables d'entrée (s'il ya des entrées). En d'autres termes, un algorithme est une procédure d'étape par étape pour résoudre un problème donné.

Analyse ascendante: C'est une technique de conception où un programme est divisé en un certain nombre de tâches plus petites et plus simples qui peuvent être abordées séparément. Dans cette technique, des sous programmes sont écrits et testés en premier avant que le programme principal ne soit écrit.

Design: le design est le développement des alternatives du problème et des modèles pour représenter le problème et la solution. C'est un processus clair à travers lequel les besoins des utilisateurs sont transformés en une représentation compréhensible par la machine.

Implémentation: Il s'agit du codage et du test du programme

Organigramme: C'est une technique normalisée pour représenter graphiquement des schémas avec un ensemble de symboles normalisés qui représentent la logique du programme.

Modèle: est une abstraction de la réalité.

Modularisation: entraîne le morcellement d'un vaste programme en sous-programmes qui sont gérables de par la taille en termes de complexité de la logique et du nombre d'instructions.

Module: est une partie d'un programme qui remplit une fonction distincte

Planning: c'est le niveau de base du développement d'un logiciel où le problème est défini et priorisé.

Programmation structurée: C'est un paradigme de programmation qui supporte la modularité en morcelant les fonctions dans des modules triviaux.

Déploiement: le déploiement entraîne l'installation du programme dans l'environnement de l'utilisateur et la formation des utilisateurs.

Chemin logique: C'est une séquence d'instructions ou déclarations également connue sous le nom d'algorithme.

Maintenance: implique d'effectuer des changements sur le programme après l'installation.

Problème: est une situation indésirable qui empêche les utilisateurs de réaliser pleinement leurs objectifs. Un problème et la nécessité de le résoudre découle d'une volonté de transformer la situation actuelle en une autre situation plus désirée (Programming and PROB solving using C par ISRD).

Processus de Développement de Programme: est un ensemble normalisé de mesures utilisées pour fournir une approche logique, le bon sens (ou processus) pour résoudre un problème difficile avec une application informatique.

Pseudocode: est une manière générique de décrire un algorithme sans une utilisation de toute syntaxe spécifique à un langage de programmation. C'est la représentation d'une solution en utilisant l'anglais non standard comme un langage de programmation. Il utilise des déclarations/mots et expressions de l'anglais courant que ce qui est généralement trouvé dans la syntaxe des langages de programmation.

Logique de programme: est utilisée par le programmeur pour modéliser les instructions du langage de programmation effectuées par l'ordinateur lorsque le programme est exécuté.

Déclarations du Langage de programmation: ce sont des déclarations basées sur l'anglais qui, lorsqu'elles sont exécutées dans la séquence correcte peut demander à l'ordinateur d'effectuer une série de tâches. Elles sont utilisées pour mettre en œuvre une logique de programme par l'envoi d'instructions au système d'exploitation.

Analyse des besoins: C'est la deuxième phase de développement d'un logiciel après la planification. Il s'agit de rechercher les spécifications du problème par le biais de questionnaires, des entretiens et d'observations entre autres méthodes.

Cycle de Vie de Développement d'un Logiciel: C'est la méthodologie ou processus utilisé par les personnes qui travaillent dans la technologie de l'information pour résoudre des problèmes en fournissant une solution efficace et efficiente.

Analyse descendante: C'est une technique de conception où un programme est divisé en un certain nombre de tâches plus petites et plus simples qui peuvent être abordés séparément. Dans cette technique, le programme principal est écrit et testé avant que les sous-programmes ne soient écrits. Ainsi, il fait voir en premier la solution dans un cadre plus large..

Activités d'apprentissage

Activité 2.1 - Résolution de problèmes

Introduction

Différentes étapes sont suivies afin d'arriver à une solution ou développer un logiciel. Le premier niveau de résolution de problèmes comprend la définition du problème qui implique la compréhension de la nature du problème à portée de main. Ce niveau de résolution de problèmes n'aborde pas comment résoudre le problème, mais nous dit ce que le problème est vraiment. Il s'agit de comprendre les entrées, les opérations et la sortie attendue du programme. Le schéma suivant illustre les étapes suivies lors de la résolution d'un problème.

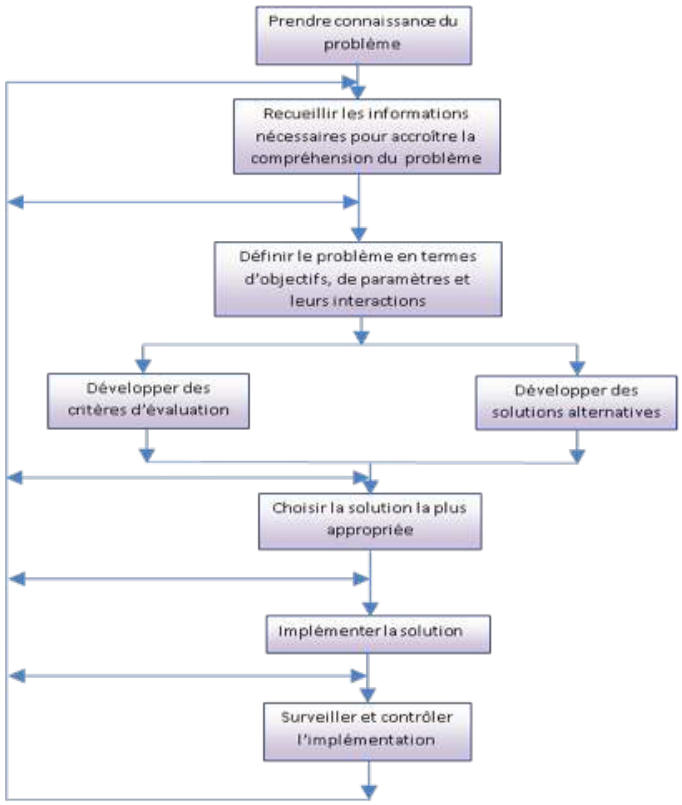


Figure 2.1 : Les étapes de résolution de problèmes (source: Programming and PROB solving using C écrit par ISRD)

"Il n'y a pas de livres de cuisine sur les stratégies pour remplacer l'intelligence, l'expérience et le bon goût de la programmation", Harsha Priya, R. Ranjeet.

Détails de l'activité

Par conséquent, dans cette activité, vous devez lire les sujets sur ce que sont vraiment les problèmes de programmation, pourquoi les ordinateurs et autres systèmes numériques sont utilisés pour résoudre les problèmes, la définition de problème, les similitudes entre les problèmes, les stratégies de résolution de problèmes et les étapes de résolution d'un problème informatique. Les lectures peuvent être obtenues à partir de (Programming and Prob solving using C écrit par ISRD, Programming and problem Solving Through C Language écrit par Harsha Priya, R. Ranjeet, et Problem solving and program design in C écrit par Jeri R. Hanly & Elliot B. Koffman) et de la liste des livres fournis dans la section référence pour cette unité. Les apprenants devraient également compléter les lectures en faisant des recherches individuelles afin d'améliorer leur compréhension. Dans cette activité, vous êtes tenus de rédiger des notes brèves sur :

- Pourquoi un ordinateur est-il un outil nécessaire pour la résolution des problèmes.
- Les stratégies de résolution de problème et leurs avantages dans la programmation.
- Processus de développement de logiciel.
- Similitudes entre les problèmes qui peuvent être résolus en utilisant un ordinateur.

Conclusion

Cette activité est destinée à aider l'apprenant à acquérir une meilleure compréhension de ce que sont les problèmes en programmation, connaître les stratégies et les étapes de la conception d'une solution et les similarités entre les problèmes.

Évaluation

1. Qu'est-ce un problème?
2. Qu'entendez-vous par le terme définition de problème?
3. Ordinateurs et autres appareils numériques tels que les téléphones mobiles sont utilisés pour résoudre des problèmes. Donnez les raisons pour lesquelles ces dispositifs sont utilisés comme outils de résolution de problèmes.
4. Pourquoi devriez-vous appliquer des stratégies dans la résolution de problèmes? et quelles sont ces stratégies? DiscuteLe développement d'un programme entraîne quelques étapes et l'une des étapes implique la modélisation de la logique. Cependant, il est possible d'arriver à une solution sans adhérer à ces étapes. Pourquoi est-il nécessaire d'appliquer le processus de développement des logiciels? Expliquez
5. La définition du problème transmet les entrées, les sorties et les opérations du programme. Discutez.
6. Les similitudes entre les problèmes peuvent inclure la vitesse de traitement, le volume de travail, la fiabilité, la précision, la sécurité, le coût, la recherche et les manipulations des chaînes de caractères. Discutez de ces similitudes.

Activité 2.2 - Conception de programme

Introduction

La conception du programme implique comment résoudre le problème et est une phase de résolution de problème qui vient après la définition du problème. Il s'agit de fournir des solutions aux problèmes en appliquant différentes techniques de conception, y compris la modélisation. Ces techniques incluent le morcellement d'un programme en des sous-programmes. Cet aspect de diviser un programme en sous-programmes conduit à la modularisation. La modularisation d'autre part, est l'un des éléments de base de la programmation structurée.

Algorithmes

Les algorithmes sont un ensemble d'étapes générées par le programmeur pour l'aider à résoudre un problème particulier. Parce qu'un algorithme est utilisé comme un outil de conception, il doit donc être fini, sans ambiguïté, simple (définitive) et précise, complète, efficace et permettre l'entrée et la sortie des données. Les algorithmes sont indépendants du langage de programmation et peuvent être développés en utilisant trois grandes constructions de programmation qui comprennent :

La construction séquentielle où les étapes sont écrites dans un ordre particulier, par exemple d'abord, l'étape A est exécutée puis B et C vient après B etc. Le flux de contrôle se fait étape par étape.

Par exemple :

```
Lire a, b

sum= a + b

Afficher sum
```

La deuxième construction est conditionnelle. Cela signifie qu'à un certain point dans l'exécution des étapes, et pour réaliser une tâche, il est nécessaire de se brancher ou de prendre une décision en évaluant une condition.

Par exemple :

```
Lire a, b

sum = a + b

si (sum > d)

    alors Afficher sum

Sinon

Afficher d
```

L'autre construction est la boucle. Dans cette construction, une étape est répétée un nombre de fois avant l'exécution de l'étape suivante ou d'autres étapes en fonction d'une condition. Lorsque la condition devient fausse, la boucle est quittée et d'autres étapes sont alors exécutées.

Par exemple:

```
Lire a, b

sum = a + b

Tant que (a<b)

    Afficher a

Lire next a, b
```

répéter l'évaluation

Rappelez-vous, nous avons défini un algorithme comme un bel ensemble d'instructions qui signifie qu'il a un début et une fin. Un exemple d'algorithme qui additionne deux nombres.

début

valeurs d'entrée pour A et B

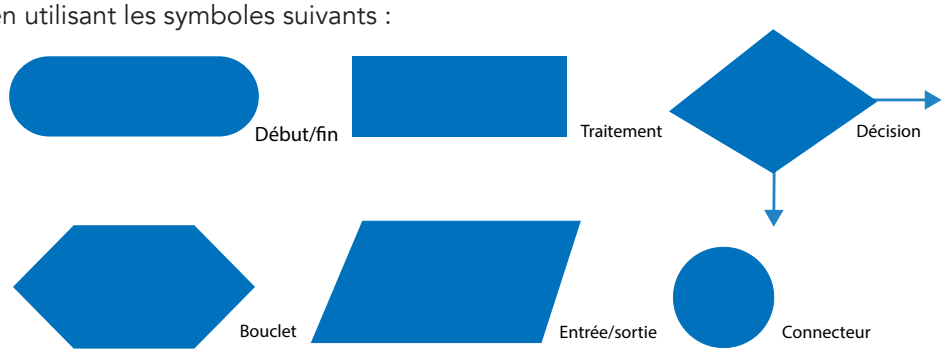
ajouter a à b et le stocker dans total

afficher la valeur de la somme

fin

Organigrammes

L'organigramme est une représentation schématique d'un algorithme. Un organigramme est conçu en utilisant les symboles suivants :



Maintenant, nous pouvons représenter notre algorithme (ci-dessus) en utilisant un organigramme.

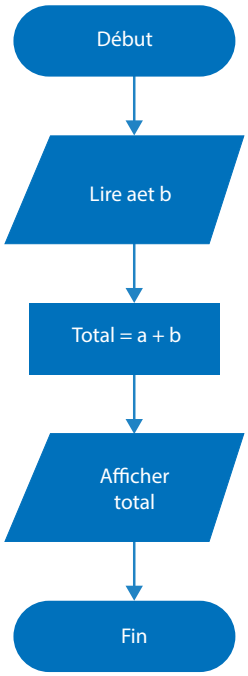


Figure 2.2 : Exemple d'organigramme

Technique de conception descendante (top-down)

Cette stratégie de conception élimine la complexité de gestion d'un gros problème en les décomposant en sous-modules pouvant être facilement compris. Le processus de division d'une tâche en sous-tâche est répété jusqu'à ce qu'on atteigne une tâche facile à mettre en œuvre. La figure ci-dessous montre comment la technique descendante peut être utilisée.

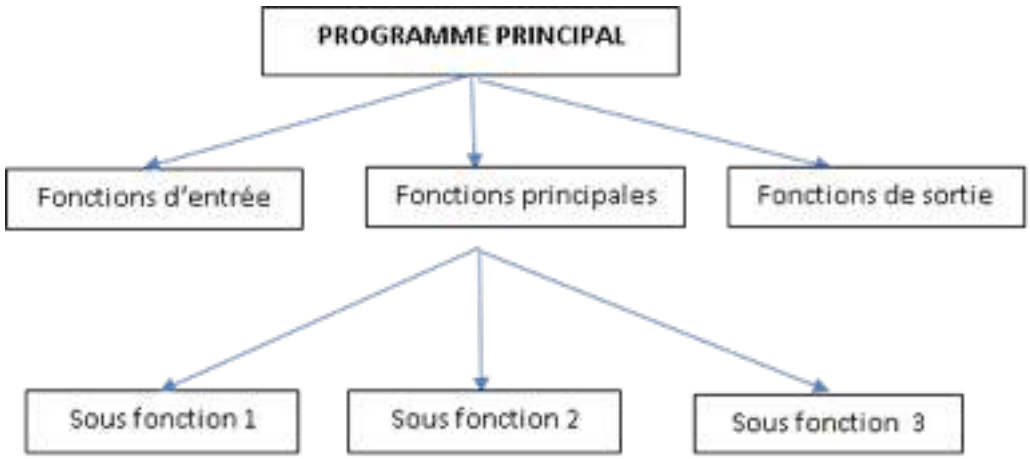


Figure 2.3 : Technique de conception top-down (source: Harsha Priya, R. Ranjeet)

Avantages

- Le principal avantage de l'approche top-down est qu'il permet dans un premier temps de voir le programme dans sa globalité.
- Pour écrire les modules de haut niveau, on n'a pas besoin de se soucier des petits détails.
- Les modules de niveau inférieur sont conçus en ayant à l'esprit les modules de niveau supérieur et on sait déjà dans les moindres détails, les objectifs à atteindre. Ce qui permet de garantir la compatibilité entre des niveaux élevés et les niveaux inférieurs.
- L'approche top-down convient aux problèmes de grande taille.

Inconvénients

- La solution offre une couverture limitée dans les premières phases
- On ne se rend pas vite compte des avancés du projet.
- Elle engendre la création d'outils particuliers et différents même si les problèmes traités sont similaires dans les différentes branches.
- etc...

Lire programming and problem solving through c language écrit par Harsha Priya, R. Ranjeet, et également, Programming and PROB solving using C écrit par ISRD pour comprendre les avantages et les inconvénients de la technique de conception top-down.

Technique de conception ascendante (bottom-up)

Ici, les sous-programmes les plus élémentaires sont écrits et plus tard utilisés pour faire des sous-routines sophistiquées. Pourquoi cette approche n'est pas très recommandée? C'est comme l'inversion de la technique de conception descendante, voir la figure ci-dessous.

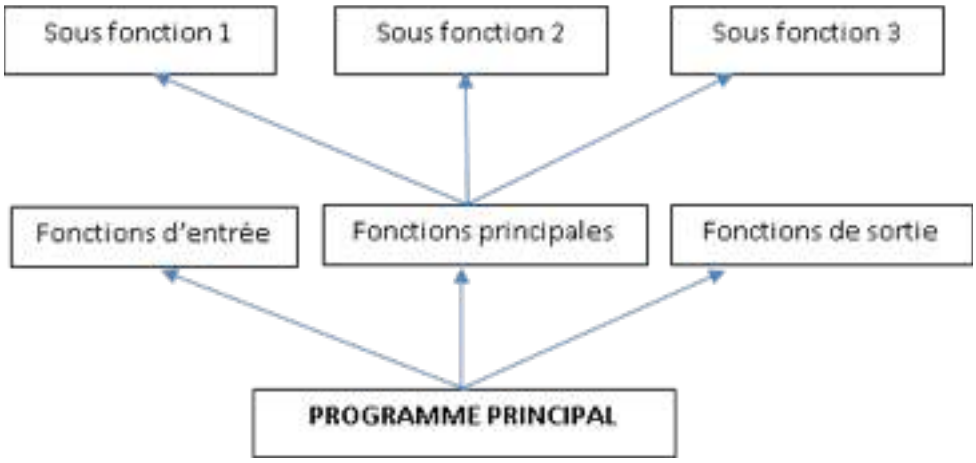


Figure 2.4 : Technique de conception bottom-up (source: Harsha Priya, R. Ranjeet)

Avantages

- L'approche bottom-up améliore la lisibilité du code.
- Permet la réutilisation du code.
- L'approche bottom-up est aussi simple car il n'est pas nécessaire de créer des talons pour les modules de niveau supérieur. En effet, les talons sont écrits et testés avant de progresser.
- Facilite le prototypage.
- L'approche bottom-up est mieux adaptée aux petits problèmes.
- etc...

Inconvénients

- On n'a pas une vue globale du programme au premier abord.
- Lorsqu'on écrit les modules de niveau inférieur, on n'a pas à l'esprit ceux de niveau supérieur. Cela peut conduire à une mauvaise connexion entre les modules de niveau supérieur et ceux de niveau inférieur lors de l'intégration des différents modules.
- Cette approche rend plus difficile la surveillance et la gestion du projet.

Lire programming and problem solving through c language écrit par Harsha Priya, R. Ranjeet, et également, Programming and PROB solving using C écrit par ISRD pour comprendre les avantages et les inconvénients de la technique de conception bottom-up.

Vue d'ensemble des techniques de conception de la programmation

Programmation Linéaire : Cela nécessite une programmation de manière séquentielle. La prise de décision et constructions d'itérations ne sont pas inclus. C'est une traduction directe d'un algorithme séquentiel.

Programmation structurée : Cette technique de conception de la programmation utilise trois grandes constructions de programmation : séquentielle, les boucles et les structures conditionnelles. Il englobe également l'utilisation des techniques de résolution de problèmes top-down (expliqué ci-dessus). Par conséquent, les programmes écrits en utilisant cette approche peuvent être facilement compris, corrigés (débogués), réutilisables, etc. Lire Programming and PROB solving using C écrit par ISRD pour comprendre les avantages et les éléments de la programmation structurée.

La conception modulaire des programmes : Les techniques descendantes conduisent à la notion de modularisation. Les programmes peuvent être logiquement séparés en : l'initialisation, l'entrée, la validation des données d'entrée, le traitement, la sortie, la gestion des erreurs et la procédure de fermeture.

Lire Programming and PROB solving using C écrit par ISRD et comprendre les attributs de base d'un module, de contrôler les relations entre les modules, la communication entre les modules et les exigences de conception du module.

Détails de l'activité

Ainsi, vous devez lire des sujets sur les techniques de conception qui comprennent les techniques de conception ascendante et descendante; leurs avantages et leurs inconvénients et où chaque technique est plus applicable. En outre, lire sur les algorithmes; les caractéristiques d'un bon algorithme, la logique de l'algorithme (flux séquentielle, conditionnelle et répétitive), la façon de tester un algorithme en utilisant la technique de vérification appelée desk checking et la technique appelée walkthrough, la différence entre un algorithme et un pseudo-code, les avantages de l'utilisation d'un pseudo-code sur les organigrammes et vice versa et la construction d'organigramme; y compris les avantages et les inconvénients de l'utilisation des organigrammes, les règles de conception d'un organigramme. Les techniques de programmation qui comprennent : la programmation linéaire, la programmation structurée (avantages et constructions de la programmation structurée), la conception modulaire des programmes. Les lectures peuvent être obtenues dans la liste des livres fournis dans la section référence pour cette unité et des informations connexes et pertinentes sur Internet. Par la suite, écrire de brèves notes pour :

- Décrire les techniques de conception de descendante et ascendante.
- Décrire le processus de développement d'un logiciel.
- Décrire les techniques de programmation.

Activité pratique

- En utilisant un crayon and du papier ou un ordinateur, construire un organigramme pour l'un des deux problèmes listé dans l'activité 1.
- Ecrire un algorithme pour l'organigramme que vous avez construit ci-dessus
- Testez votre algorithme utilisant la technique de vérification desk checking

Conclusion

Cette activité équipe l'apprenant des techniques de conception de solution. L'apprenant acquiert également une expérience pratique sur la façon de concevoir des solutions en utilisant des organigrammes et des algorithmes/pseudocodes

Évaluation

1. Qu'entendez-vous par le mot conception de programme?
2. Nommer et décrire deux techniques de résolution de problèmes.
3. Décrire la différence entre les deux techniques de résolution de problèmes mentionnées ci-dessus (2)
4. Les algorithmes peuvent être présentés par l'utilisation de pseudo-code ou un organigramme. Quelle est la différence entre un algorithme et un pseudo-code.
5. Le pseudocode est indépendant d'un langage. Qu'est-ce que cela veut dire?
6. Quelle est la signification de la documentation dans le processus de résolution de problème?
7. Les algorithmes qui sont développés en utilisant trois constructions de base sont plus faciles à suivre. Décrire ces constructions.
8. Décrire les caractéristiques d'un bon algorithme
9. Quels sont les avantages des organigrammes sur les algorithmes
10. Nommez les règles pour construire un organigramme
11. La conception des programmes définit la structure du code du programme. Décrire trois techniques de programmation
12. Quels sont certains des avantages de la programmation structurée
13. Décrivez trois éléments de la programmation structurée
14. Un algorithme peut être testé en utilisant la technique de vérification appelée desk checking et la technique appelée walkthrough. Décrire ces approches de tests.

Activité 2.3 - Activité de recherche en groupe

Introduction

Cette activité consiste à faire en groupes ou en collaboration de la recherche sur certains aspects de la programmation. L'activité nécessite que l'apprenant obtienne à partir de programmeurs pratiquants le genre de problèmes qu'ils résolvent, les stratégies qu'ils appliquent et les étapes suivies pour parvenir à une solution et les opérations communes à tous les programmes. Ils sont tenus de connaître le réalisme des différentes techniques de résolution de problèmes et leur signification.

Détails de l'activité

Dans cette activité, vous devez :

- Vous renseigner auprès d'un développeur sur le genre de problèmes qu'il résout de façon quotidienne. Utilisez les exemples réunis pour vous aider à réfléchir sur un programme beaucoup plus vaste (et pas trop grand) et en appliquant les concepts appris dans cette unité.
- Préciser ou formuler le problème et le définir en montrant les personnes impliquées, les entrées, les opérations et la sortie.
- Concevoir la solution au problème en utilisant

  (1). la technique descendante et

  (2). la technique ascendante.

- Utiliser les organigrammes et les pseudocodes / algorithme pour représenter la solution.
- Utiliser la technique de vérification desk checking pour tester votre pseudocode / algorithme.
- Collaborer avec un autre groupe pour échanger vos solutions conçues (Algorithmes) puis effectuer une vérification par les pairs (walkthrough).

Conclusion

Grâce à cette activité, l'apprenant sera en mesure d'obtenir plus de connaissances sur la résolution de problèmes avec l'ordinateur. En collaborant avec d'autres, l'apprenant sera capable d'améliorer ses compétences sur les applications des concepts de résolution de problèmes.

Évaluation

1. Nommer les opérations qui sont communes à toutes les solutions
2. Outre l'entrée, les opérations et la sortie, quelles autres caractéristiques considérez-vous lors de la définition d'un problème?
3. Décrire une propriété d'un problème bien posé
4. Quels sont les avantages de la technique descendante sur la technique ascendante
5. Décrivez les résultats que vous avez obtenus à partir de la technique de vérification desk checking.
6. Quelle est la signification de tester un algorithme?
7. Quels bénéfices avez-vous eu en collaborant avec vos coéquipiers et les membres d'une autre équipe?

Résumé de l'unité

Cette unité a discuté des stratégies de résolution de problème avec l'ordinateur, les étapes suivies lors de l'élaboration d'un programme, les techniques de conception qui comprennent l'approche descendante et l'approche ascendante, les algorithmes et les deux façons de présenter un algorithme; le pseudocode et le organigramme. Il a également été examiné les moyens de tester un algorithme; vérification desk checking et technique walkthrough (contrôle par les pairs), et un aperçu des techniques de programmation.

Évaluation de l'unité

Vérifiez votre compréhension!

Questions

1. Un organigramme est _____ (2 points)
2. Nous avons besoin d'ordinateurs pour résoudre les problèmes en raison de : (2 points)
   a. la vitesse dans l'exécution des tâches de calcul
   b. moins sujettes aux erreurs
   c. réduction de la paperasse
   d. polyvalence des opérations effectuées
   e. tout ce qui précède
   f. a, b et c

3. Considérons le système utilisé pour contrôler l'arme nucléaire. L'exposition des êtres humains à cet environnement dangereux n'est pas conseillée. De plus la précision de l'opération en cause est obligatoire comme le système est très critique par nature. Quels sont les avantages d'un système informatique dans un tel scénario. (2 points)

   a. précision de l'opération

   b. sécurité du genre humain

   c. disponibilité du système

   d. aucune de ces réponses

   e. tout ce qui précède

4. La définition du problème implique (2 points)

   a. Identification des personnes clés impliquées

   b. Identification des entrées

   c. Spécification de la sortie de la solution

   d. Spécification des opérations du programme

   e. b, c et d

   f. tout ce qui précède

5. Simplicité, clarté et élégance sont les maîtres mots des bons programmes, mais diverses questions de mise en œuvre doivent être prises en considération lors de la conception d'une solution. Lesquels ? (2 points)

   a. minimiser les besoins en mémoire

   b. maximiser la lisibilité de la sortie

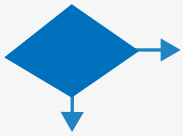
   c. maximiser la lisibilité du texte d'origine code source

   e. réduire le temps de développement

   f. a et c

   g. tous

6. Faites correspondre les notations d'organigramme suivantes avec leurs fonctions et noms corrects. (6 points)

	Notation	Fonction	Nom
a.		reliant deux étapes	traitement
b.		représentent une condition	bouclage
c.		prend des valeurs d'entrée/sortie	connecteur
d.		Effectuer calculs/traitement	entrée / sortie
e.		Boucle pour une série d'étapes jusqu'à ce que la condition soit satisfaite	décision
f.		Indiquer le début/fin d'un processus	début / fin

Instructions

Le but de cette évaluation est de vérifier les progrès de l'apprenant en déterminant ce que l'apprenant a appris dans cette unité. Les questions couvrent tout ce qui a été présenté dans cette unité afin d'évaluer la compréhension globale. Répondez attentivement et si votre score tombe

- en dessous de 40%, refaire les lectures.
- entre 40% et 60%, refaire les lectures sur votre zone de faiblesse
- supérieur à 60%, vous avez une quantité importante de connaissances

Système de notation

Devoirs 20%

Test d'évaluation10%

Examen final 70%

Réponses

1. Un organigramme est une représentation schématique d'un algorithme
2. e
3. e
4. f
5. g
6. a. indiquer le début et la fin d'un processus *début / fin*

   b. effectuer des calculs et le traitement *traitement*

   c. représentent une condition prise de décision

   d. boucle pour une série d'étapes jusqu'à ce que la condition soit satisfaite *bouclage*

   e. prenant des valeurs d'entrée/sortie *entrée / sortie*

   f. reliant deux étapes *connecteur*

Lectures et autres ressources

- Programming and problem solving through "C" Language. Harsha Priya, & R. Ranjeet. Firewall Media, 2006.
- Programming & Prob solving using C. ISRD, Tata McGraw-Hill Education
- Fundamentals of Programming: with Object Oriented Programming. Gary Marrer. 2009. Gary Marrel.
- Programming Logic and design, Comprehensive. Joyce Farrell, 2012. 7th edition. Cengage Learning.
- Problem solving and program design in C. Jeri R. Hanly & Elliot B. Koffman, 6, illustrated, Addison-wesly, 2009.

Unité 3. Programmation en langage C

Introduction de l'unité

Les concepts de la programmation peuvent être différents selon les langages, mais quelques instructions de base apparaissent dans presque tous les langages. Ces instructions incluent les instructions d'entrée qui sont utilisés pour obtenir des données à partir du clavier, un fichier, ou tout autre dispositif; les instructions de sortie utilisées pour afficher des informations à l'écran ou envoyer des informations dans un fichier ou tout autre dispositif; les instructions arithmétiques effectuant des opérations arithmétiques de base comme l'addition et la multiplication; les instructions conditionnelles utilisées pour vérifier certaines conditions et exécuter la séquence appropriée des déclarations et les instructions de répétition effectuant une action à plusieurs reprises, le plus souvent avec une certaine variation. Par conséquent, cette unité traite les concepts de programmation comme les variables, les types de données, les opérateurs, les structures de contrôle, les tableaux et les procédures ou fonctions. Les procédures ou fonctions renforceront le concept de décomposition d'un problème en problèmes plus petits problèmes (techniques de conception de haut en bas et / ou de bas en haut). Des programmes simples seront écrits en langage C pour renforcer toutes les notions apprises dans l'unité précédente.

Objectifs de l'unité

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

- Télécharger et installer un programme.
- Distinguer les différents types de données.
- Fournir des solutions en écrivant des programmes simples en C.

Termes clés

Type de données: types de données définissent la manière dont les valeurs et les choix de valeurs sont représentées dans un système

Variables: ce sont des noms donnés à la zone de mémoire de stockage. La valeur d'une variable est modifiée pendant l'exécution du programme.

Une constante: Représente une valeur stockée dans la mémoire et qui ne peut pas changer.

déclaration de variable: déclare le type de la variable et alloue de la mémoire pour cette variable.

Un identifiant: C'est un nom de variable

Application Console: Une application de la console est un programme informatique conçu pour être utilisé par une interface de l'ordinateur en mode texte, comme un terminal texte , une interface en ligne de commande de certains systèmes d'exploitation (Unix , DOS , etc).

Environnement de développement intégré (IDE): Un environnement de développement intégré (IDE) est une application logicielle qui fournit des installations complètes pour les programmeurs informatiques de développement de logiciels. code source: C'est une programme de l'ordinateur écrit dans un langage de haut niveau qui est converti en code objet ou code machine par un compilateur.

Types de données dérivées: Ils sont définis en fonction d'autres types de données, appelés types de base. Les types dérivés peuvent avoir des attributs, et peuvent avoir des éléments ou un contenu mixte (http://msdn.microsoft.com).

Types de données primitifs: les types de données primitifs ne sont pas définis en fonction d'autres types de données, car ils constituent la base de tous les autres types. Ils ne peuvent pas avoir des contenus d'éléments ou des attributs (http://msdn.microsoft.com).

Constante String: c'est une chaîne de caractères entre guillemets. (Fondamentale)

Un opérateur: C'est un symbole qui permet à l'utilisateur de demander à l'ordinateur de faire un certain nombre de calcul mathématique ou logique. (Fondamentale)

Types de données définis par l'utilisateur: Ils sont définis par l'utilisateur en fonction des types de données existants.

Éditeur de code source: Un éditeur de code source est un éditeur de programme basé sur le texte et spécialement conçu pour l'édition de code source des programmes informatiques par les programmeurs . Il peut être une application autonome ou il peut être intégré dans un environnement intégré de développement (IDE) ou navigateur Web.

(http://en.wikipedia.org/wiki/Source_code_editor)

Fichiers d'en-tête: les fichiers d'en-tête contiennent des définitions de fonctions et de variables qui peuvent être incorporés dans n'importe quel programme C en utilisant l'instruction de préprocesseur, #include (http://gd.tuwien.ac.at/languages/c/programming-bbrown/c_011.htm).

Bibliothèque: Une bibliothèque est une collection de programmes (généralement) précompilés, réutilisables qu'un programmeur peut «appeler» lors de l'écriture de son code afin qu'il n'ait pas besoin de les réécrire (searchsqlserver.techtarget.com/definition/library).
Une bibliothèque en C est un groupe de fonctions et de déclarations, exposées pour une utilisation par d'autres programmes. (http://en.wikibooks.org/wiki/C_Programming/Libraries).

Un commentaire: Un "commentaire" est une séquence de caractères commençant par une combinaison de slash/astérisque (/*) et se terminant par un astérisque/slash (*/) qui est considérée comme un seul caractère d'espace blanc par le compilateur et est ignorée. Il existe également des commentaires en ligne précédés par deux barres obliques (//) (http: // msdn.microsoft.com/en-us/library/wfwda74e.aspx).

Une expression: une expression est une combinaison de variables, des constantes, et d'opérateurs conformément à la syntaxe du langage (livre par ISRD)

Type de casting: La notion transtypage en langage C est utilisée pour modifier le type d'une variable. Le nouveau type de données doit être mentionné avant le nom de variable entre parenthèses (http://fresh2refresh.com/c/c-type-casting/).

Portée des règles: C'est une région du programme où une variable définie peut avoir son existence et au-delà de laquelle cette variable ne peut être accessible. On distingue les variables ovales, les variables globales et les paramètres formels (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm).

Paramètres formels: Ce sont des paramètres qui agissent comme des espaces réservés pour les paramètres réels.

Paramètres effectifs: Ce sont les paramètres (ou valeurs) transmis aux paramètres réels de la fonction d'appel (environnement).

Les variables locales: Les variables qui sont déclarées dans une fonction ou d'un bloc sont appelées variables locales. Ils peuvent être utilisés que par des déclarations qui sont à l'intérieur de cette fonction ou de bloc de code. (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm)

Les variables globales: Les variables globales sont définies en dehors d'une fonction, généralement en haut dans le programme. Les variables globales gardent leur valeur pendant toute la durée d'exécution de votre programme et peuvent être consultée à l'intérieur de l'une des fonctions. (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm)

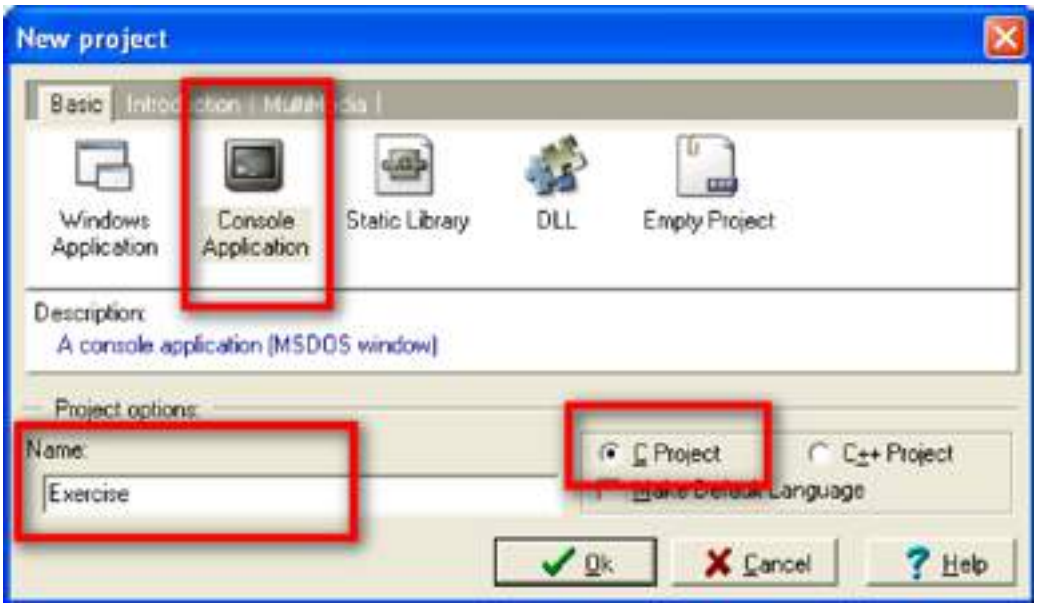
Activités d'apprentissage

Activité 3.1 - Types de données, variables et opérateurs

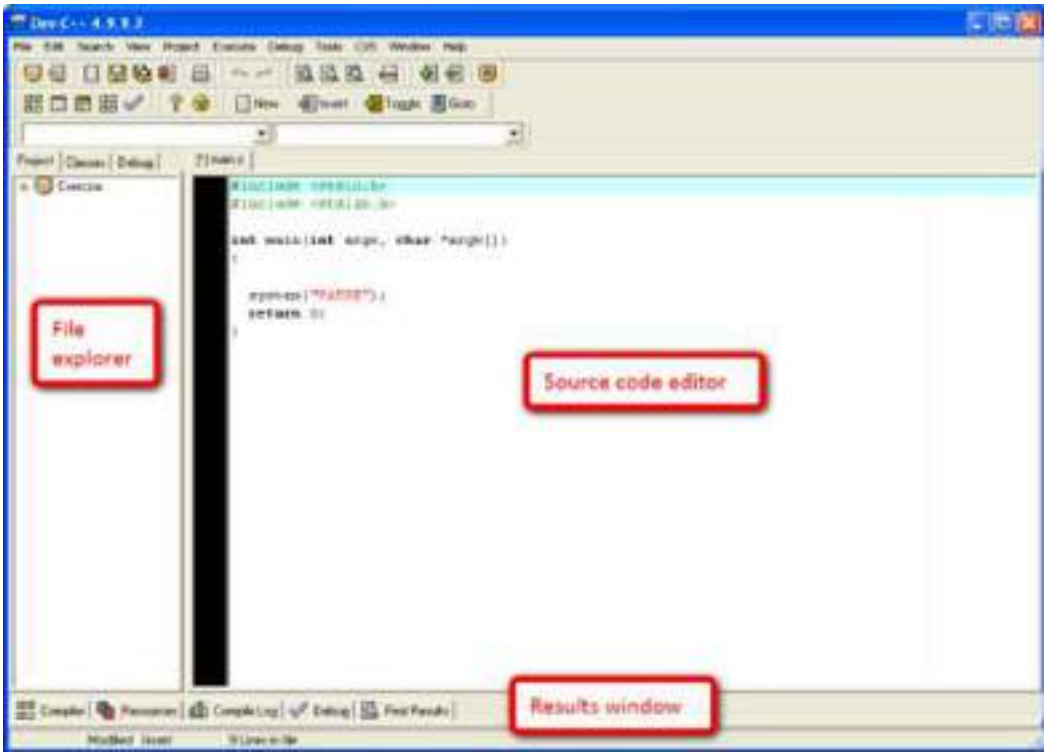
Introduction

Afin de discuter et d'illustrer ces concepts à l'aide d'un programme C, nous devons d'abord commencer par nous familiariser avec l'environnement de développement (aperçu du langage C). Il y a de bonnes références qui constituent un bon point de départ pour la programmation C, y compris les références fournies pour cette unité. programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet, Fundamentals of programming languages by Dipali P. Baviskar and Programming and PROB solving using C. by ISRD. Beaucoup de compilateurs C existent sur le marché et ont été conçus ou développés par différents groupes de développeurs de programmes. Par conséquent, la préférence d'un compilateur est laissée à vous (apprenant). La préférence de ce cours est le compilateur Dev C++. Vous pouvez le télécharger, à partir de http://liquidtelecom.dl.sourceforge.net (Dev-cpp 5.8.3 TDM-gcc 4.8.1 setup.exe) ou la dernière version mise à jour qui peut fonctionner sur la dernière version de Windows. Si vous n'êtes pas en mesure de le récupérer, demandez à votre instructeur de vous aider. Le compilateur lit c ++ et cela ne devrait pas être un souci pour vous car vous allez apprendre comment compiler un programme C en renommant les fichiers pour avoir l'extension C (.c).

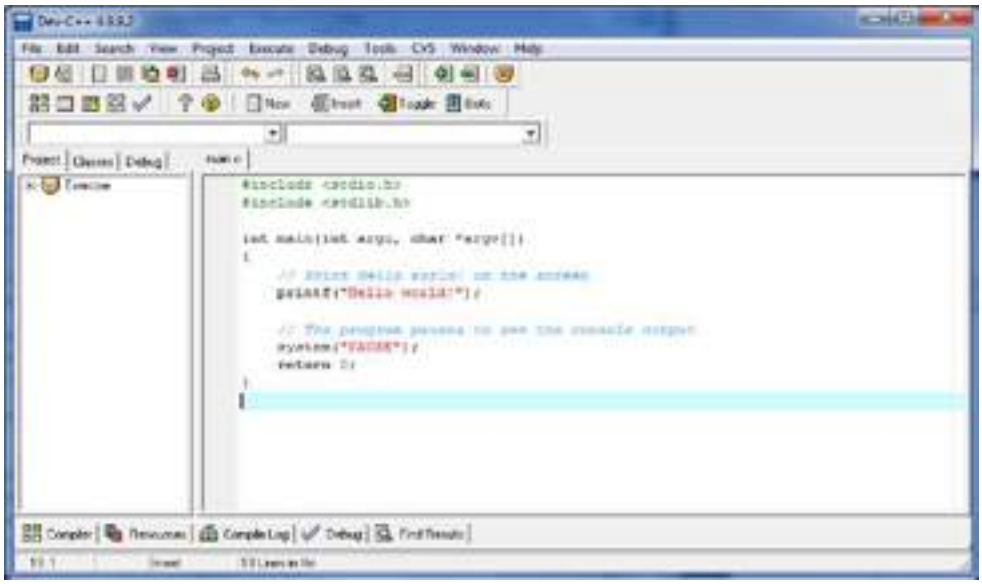
Avant de créer le fichier ducode source, il est nécessaire de créer un projet (Fichier> Nouveau> Projet) et les options comprennent l'application de la console, le nom du projet et projet C pour le langage C.



L'activité suivante consiste à sélectionner le dossier pour stocker les fichiers, dans la fenêtre suivante. Après avoir indiqué le dossier où le fichier de configuration du projet (.dev) sera sauvegardé, l'IDE génère un fichier source de base (par défaut, main.c). La fenêtre IDE comprend trois sous-fenêtres: l'explorateur de fichier du projet, les onglets de résultat, et l'éditeur de code source. Ces fenêtres peuvent être redimensionnées et minimisés.



L'instruction system("PAUSE"); est automatiquement inclus pour mettre en pause l'exécution du programme avant de fermer la fenêtre pour permettre de voir les résultats du programme. La fenêtre Explorateur de Fichiers (Files Explorer) affiche le nom du projet et les fichiers inclus. L'onglet Projet (Project tab) contient généralement un seul fichier avec le code source du programme. Dans ce volet, nous pouvons trouver deux onglets supplémentaires: Classes et Debug. L'onglet Classes affiche les fonctions du programme tandis que l'onglet Debug montre des variables visualisées durant le processus de débogage. La fenêtre des résultats (Results Windows) est utilisés pour présenter les résultats des actions de l'IDE: les erreurs de compilation, les directives de compilation, les commandes de débogage. L'éditeur de code source affiche le code du programme. Une fois le projet a été créé, nous pouvons commencer à écrire puis exécuter notre programme C. Par exemple, le programme simple, Hello world! ci-dessous.



Le procédé d'exécution d'un programme en C comporte les étapes suivantes:

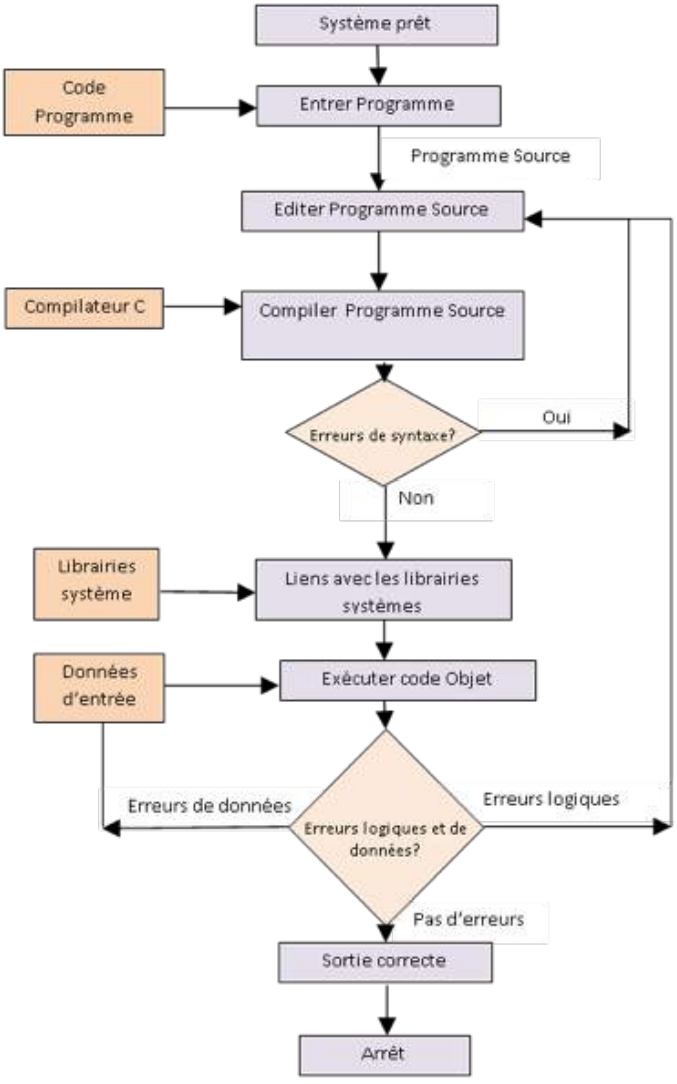
1. Création du programme
2. Compilation du programme
3. Lier le programme avec les fonctions qui sont nécessaires à partir de la bibliothèque C
4. L'exécution du programme

Remarque:

Le contenu ci-dessus a été emprunté à http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf .

Par conséquent, nous vous conseillons de visiter le site pour en savoir plus. Il introduit Dev C ++ d'une manière simple et claire (étape par étape), qui vous guidera afin que vous puissiez facilement commencé. Ne vous préoccupez pas tellement de la partie débogage car nous allons l'introduire dans l'unité suivante.

Un résumé de ces étapes peut être représenté par l'organigramme ci-après:



Processus d'exécution de source de programme C: ISRD

Création du programme: Le programme qui doit être créé doit être saisi dans un fichier. Le nom de fichier peut être composé de lettres, chiffres et caractères spéciaux, suivi de l'extension .C. Exemple de noms de fichiers valides :

Hello.c

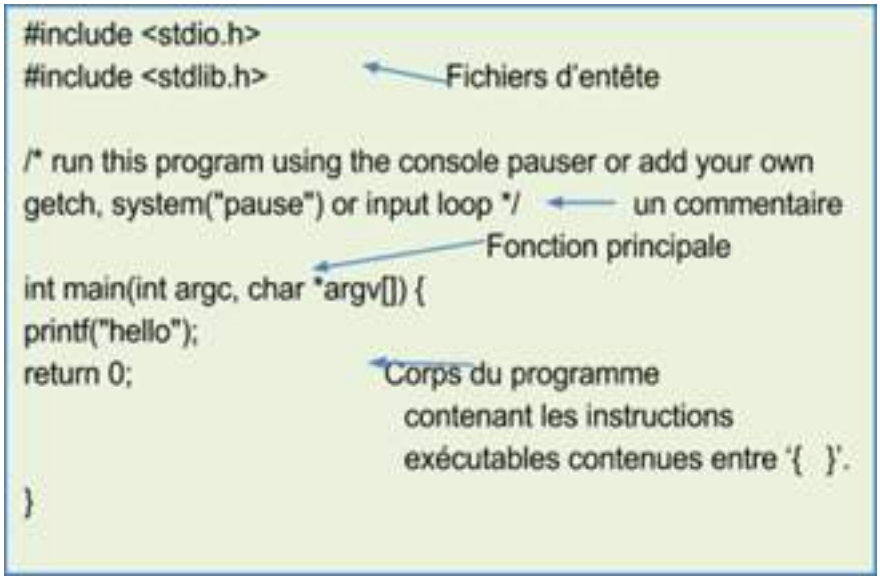
Pract1.c

Compilation et édition de lien: Pendant le processus de compilation des instructions du programme source sont convertis en une forme qui convient pour l'exécution par l'ordinateur. Le processus de traduction vérifie l'exactitude de chaque instruction et si aucune erreur n'est signalée, il génère le code objet.

Pendant l'étape reliant les autres fichiers du programme et les fonctions qui sont requis par le programme sont mis ensemble. Si des erreurs de syntaxe et de sémantique sont découvertes, elles sont répertoriés et affichées et le processus de compilation s'arrête. Les erreurs doivent être corrigées dans le programme source à l'aide d'un éditeur et la compilation se fait à nouveau.

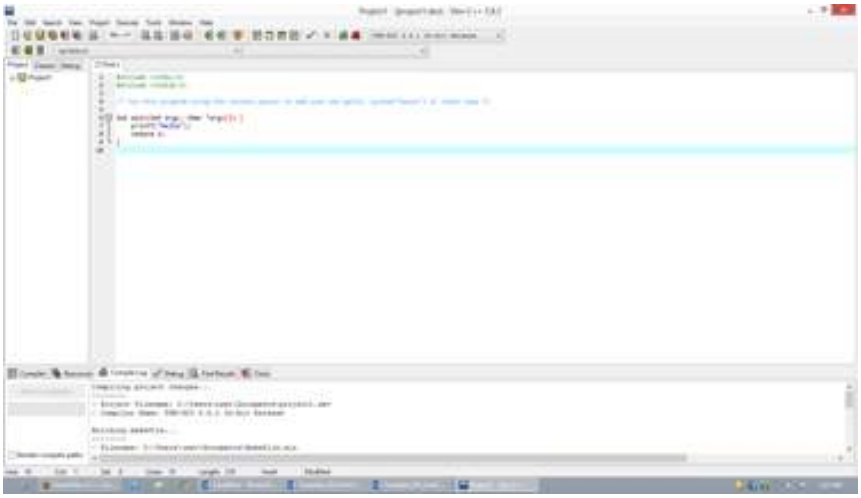
L'exécution du programme: Durant l'exécution, on charge le code objet exécutable dans la mémoire de l'ordinateur qui exécute les instructions. Pendant l'exécution, le programme peut demander certaines données par l'intermédiaire du clavier.

Un exemple décrivant des parties de base d'un programme C en utilisant Dev C ++.



Structure d'un programme C

Voir figure ci-dessous pour un programme en utilisant la version 5.6.2 de Dev C ++



Fonctions printf () et scanf ()

La fonction printf () est utilisée pour écrire des informations dans un fichier, à l'écran ou tout autre périphérique de sortie. Tandis que la fonction scanf () est utilisée pour lire les données que le programme manipule ou écrit à l'écran. Ces fonctions sont prises en charge par le fichier d'en-tête ou directive du préprocesseur <stdio.h>, ce qui signifie fichiers d'entête d'entrée et de sortie standard. Par exemple, un programme qui affiche une valeur qui est déjà affectée à une variable et une valeur (age) donnée le système par l'utilisateur. Le programme invite l'utilisateur à entrer son âge :

```
#include <stdio.h>

#include <stdlib.h>

/ * Exécuter ce programme en utilisant le system("pause") ou
une boucle d'entrée * /

int main (int argc, char * argv []) {

int y = 30; // Valeur directement affectée à la variable

int age;

printf ("Entrez votre âge \ n"); // Demander à l'utilisateur
de lire son

âge

scanf("%d",&age); // Lecture de l'âge en tapant les valeurs
sur le

                    clavier

printf("y=%d\n",y); // impression de la valeur de y

printf("age=%d",age); // impression de âge

return 0;

}
```

N'oubliez pas d'utiliser le formatage exigé lors de la lecture et de l'écriture de valeurs. En outre, utiliser l'opérateur d'adresse '&' lors de la lecture des valeurs. Utiliser la tabulation quand cela est nécessaire, pour rendre votre code plus lisible.

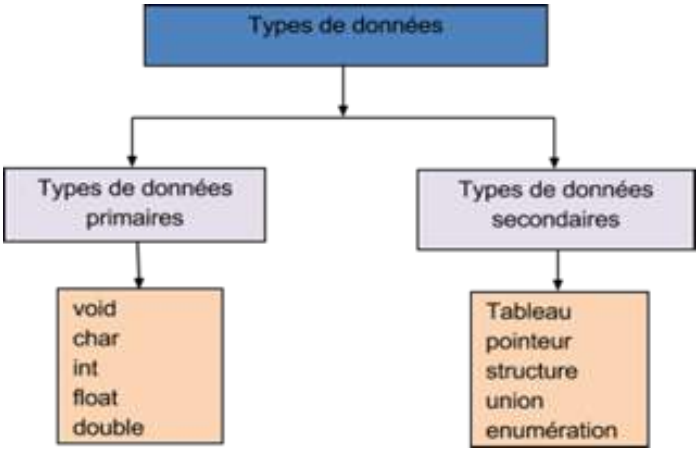
Les types de données

Un programme sauvegarde ou accepte différents types de données y compris : entiers, caractères et valeurs réelles avec les mots-clés suivants : int, char et float respectivement. Tous ceux-ci sont définis comme des types de données. Différents types de données occupent différentes tailles en mémoire. Par exemple, un caractère occupe un octet de l'espace mémoire, les nombres entiers occupent quatre octets, etc., comme indiqué dans les conditions du tableau ci-après. Mais les exigences de mémoire peuvent varier d'un compilateur à l'autre. Par conséquent, il est important de choisir le type de données correctes pour éviter le gaspillage de la ressource vitale de l'ordinateur (la mémoire).

Les types de données avec leurs besoins en mémoire et propriétés:

Type de données	Exigences de mémoire	Propriétés de valeurs de données
void	0 Octet	Rien, un objet non existant
char	1 Octet	Contient des variables de caractère comme, 'a', '&', etc.
int	2 Octet	Valeurs entières, par exemple 12, 15
float	4 Octet	Valeurs à virgule flottante; valeur unique de précision par exemple 13,7, 78,0
double	8 Octet	Valeurs à virgule flottante; double précision par exemple 17E + 21

En dehors de ces types de données, C supporte d'autres types de données connues sous le nom de types de données secondaires. La figure ci-dessous donne un résumé des types de données :



Les types de données

Les types de données primaires sont également connus en tant que types de données primitifs. Les types de données secondaires peuvent être divisés en types de données définis par l'utilisateur et en types de données dérivés. Par exemple, la structure, l'union et l'énumération sont des types de données définis par l'utilisateur tandis que les tableaux et les pointeurs sont des types de données dérivés.

Variables

Les données sont stockées dans l'emplacement de mémoire appelé variable. Cela signifie qu'un programmeur nomme un emplacement dans la mémoire et précise la quantité d'espace nécessaire pour stocker les données en définissant le type de ces données. Le nom de l'emplacement mémoire est reconnu comme un identificateur. Ce nom donne un accès convivial à l'emplacement mémoire. Par exemple, envisager une situation où le programmeur a besoin de stocker les détails d'un étudiant qui comprennent, l'âge, le sexe et les frais. L'âge est un nombre entier sans fraction. Par conséquent, int est un type de données approprié pour l'âge. Frais peut avoir des valeurs à virgule, float est donc le type approprié pour frais. Sexe peut être représenté comme un seul caractère, «F» ou «M» signifie qu'une variable de type char est appropriée. Ainsi, les identifiants et les exigences mémoire appropriées donnent lieu aux déclarations suivantes :

```
int age;
   float frais;
   char sexe;
```

Chaque langage a ses règles pour créer les noms des variables ou identificateurs. C suit les règles suivantes :

1. C est un langage sensible à la casse ainsi 'Age' et 'age' sont différents
2. Les noms de variables ne doivent pas commencer par un chiffre, par exemple 2014Frais n'est pas correcte
3. Les caractères spéciaux comme '%', '@' etc, sauf le trait de soulignement '_' ne sont pas autorisés dans un nom de variable. Par exemple -feeBalance n'est pas correcte mais fee_Balance est correcte.
4. Une variable doit commencer par un alphabet ou le caractère spécial '_'. (Juste comme une convention, il n'est pas conseillé d'utiliser des noms de variables commençant pat un soulignement '_' étant données que les fonctions de bibliothèque suivent cette convention)

Constantes

C supporte un certain nombre de constantes qui sont:

1. Constantes entières

Par exemple 234, -78, etc. composée d'un ensemble de chiffres de 0 à 9 et peut être représenté sous forme de chiffres signés en utilisant des valeurs + (positve) ou - (négatives)

2. Les constantes réelles

Par exemple -0.98, 10.78, etc. composée d'une partie décimale

3. constantes de caractères

Par exemple 'v', 'j', 'y', 'n', etc. entre deux apostrophes ('.')

4. Les constantes de chaîne

Par exemple "Jeanne", "étudiant", "maison", etc. entre deux guillemets doubles ("...").

Les constantes peuvent être définies en utilisant un opérateur d'affectation (=), ou un mot clé const ou la directive #define

Autres constantes de caractères (séquence d'échappement)

Il y a beaucoup d'autres caractères non-imprimables et C représente ces caractères en utilisant une séquence d'échappement (\). Par exemple '\n' représente une nouvelle ligne. Il met des informations sur une nouvelle ligne.

Opérateurs

Les opérateurs peuvent être utilisés directement sur des données ou sur des variables. Il existe deux grands types d'opérateurs qui comprennent : opérateurs unaires et opérateurs binaires. Les opérateurs unaires fonctionnent sur un seul (ou unique) opérande, par exemple +67, -5, ++x, --y, etc. Tandis que les opérateurs binaires fonctionnent sur deux ou plusieurs opérandes, par exemple 2 + 3 (+ est un symbole d'addition et fonctionne sur deux opérandes (valeurs)), d/x (/ est la division et fonctionne sur deux opérandes) produisant une valeur différente. Les opérateurs peuvent également être classés en différentes catégories, à savoir :

1. Les opérateurs arithmétiques

OPERATEUR	SIGNIFICATION
+	Addition
-	Soustraction
/	Division
*	Multiplication
%	modulo

2. Les opérateurs relationnels

OPERATEUR	SIGNIFICATION
==	Egal à
!=	Différent de
>	Strictement supérieur à
>=	Supérieur ou égal à
<	Strictement inférieur à
<=	Inférieur ou égal à

3. Les opérateurs logiques

OPERATEUR	SIGNIFICATION
&&	ET
||	OU
!	Non

4. L'opérateur d'affectation

Il est représenté en utilisant un opérateur '=' par exemple x = 6;

5. Les opérateurs d'incrémentation et de décrémentation

OPERATEUR	SIGNIFICATION
++	Incrémentation
--	Décrémentation

6. L'opérateur conditionnel

C'est un opérateur ternaire qui contient trois opérandes et sa syntaxe ressemble à ceci :

exp1 ? exp2 : exp3;

7. L'opérateur virgule

Il est représenté en utilisant un opérateur ','. Il est utilisé pour séparer les expressions.

8. L'opérateur sizeof

Il est utilisé pour renvoyer le nombre d'octets qu'un opérande occupe dans la mémoire. Par exemple :

```
char y;

y=sizeof(char);

printf("&d",y);


La sortie est: y = 1
```

9. Les opérateurs binaires

OPERATEUR	SIGNIFICATION
&	ET bit à bit
|	Inclusif bit à bit OU
^	Exclusif bit à bit OU
<<	Décalage à gauche
>>	Décalage à droite
~	Son compliment

Opérateurs composés

Il s'agit d'opérateurs utilisées pour calculer le contenu d'une variable, et assigner attribuer le résultat à la même variable. Ils incluent :

OPERATEUR	SIGNIFICATION
+=	Ajoute et assigner à
-=	Soustraire et assigner à
/=	Diviser et assigner à
*=	Multiplier et assigner à to
%=	Trouver le reste et assigner à

Exemple:

a += b; qui est la même chose que a = a+b. Cela signifie : additionner la valeur de a et de b et mettre le résultat dans a.

Expressions

Des valeurs doivent être affectées aux variables avant l'évaluation des expressions. Par exemple:

variable = expression;

R = ((a+b/j) * (h*b));

Transtypage

Forcer une des données d'un type particulier à retourner un autre type La syntaxe est:

(Type-désiré) exp;

Par exemple 8/7 doit retourner 1 mais on peut forcer pour obtenir une valeur réelle retournée.

```
int x;

x = (float) 8/7;

La sortie est : 1,14
```

Détails de l'activité

Vous êtes appelés à lire davantage sur printf(), scanf(), variables, constantes, séquence d'échappement et les opérateurs. L'activité inclut aussi la pratique. Après lecture, vous devrez écrire des programmes simples qui vous aideront à comprendre les principaux concepts de la programmation. Il est conseillé d'utiliser "Fundamentals of programming languages by Dipali P. Baviskar and Programming and PROB solving using C. by ISRD". Ces livres vous guideront à apprendre et à comprendre les notions relatives aux variables, constantes et operateurs. Vous pouvez également rechercher les documents pertinents sur ces sujets sur Internet. Après lecture, on vous demande de :

- Ecrire une note brève sur les types de données, les variables, les différents types de constantes et opérateurs.

Activité pratique

1. Téléchargez et installez un environnement de développement de programme C sur votre ordinateur (si vous n'en avez pas encore). Vous pouvez demander de l'aide sur cette activité.
2. Familiarisez-vous avec l'environnement en langage C, écrire un programme simple qui affiche votre nom.
3. Apprenez à créer des identifiants, à les définir et à les déclarer
   i. déclarer une variable pour stocker l'initiale de votre nom
   ii. déclarer une variable pour stocker votre âge
   iii. déclarer une variable pour stocker votre taille
   iv. déclarer une constante qui garde les intérêts (0.12)
   v. déclarer une variable qui garde un montant principal (nombre réel)
4. Écrire des programmes simples qui vous aideront à apprendre à utiliser des variables, affecter des valeurs à des variables, utiliser des opérateurs et les résultats de sortie.

   - En utilisant l'opérateur d'affectation, initialiser les variables déclarées de i à v et afficher leurs valeurs.

5. Écrire une seule instruction qui:

   - Lit un entier x
   - Lit une réel double y
   - Lit un caractère i

- Affiche l'entier x
- Affiche le réel double y
- Affiche le caractère i
- Incrémente x de 1

6. En utilisation des types de données appropriés, les identifiants/constantes et les opérateurs, écrire un programme en C qui se lit deux valeurs, leur ajoute des valeurs, donne le produit de deux valeurs, divise la somme des deux valeurs par 3, incrémente la valeur obtenue en ajoutant deux valeurs (en utilisant la méthode du préfixe), décrémente la valeur obtenue à partir du produit des deux valeurs par 2 puis affiche les résultats à l'écran.

7. Ecrivez un programme qui met en œuvre le code suivant. Montrer la sortie.
int a, b, c, d=5;

```
a=++d;

b=a++;

c=b--;

 printf("%d %d %d %d %d", a, b, ++c, d, --d);
```

Conclusion

Cette activité vous a permis d'apprendre à exécuter un programme en utilisant Dev C ++ (on recommande la dernière version). Vous avez également su définir et déclarer des variables en utilisant différents types de données. De plus, vous avez de manière appropriée testé tous les opérateurs mentionnés dans cette unité.

Évaluation

1. Définir les termes suivants;
   i. un type de données
   ii. une variable
   iii. un opérateur
   iv. un identificateur
   v. un mots-clés de programme
   vi. type de données dérivé
   vii. fichier d'en-tête
2. Nommez les règles qui sont suivies quand on veut créer avec un identifiant
3. Pourquoi est-il important de comprendre comment utiliser les types de données?

4. Le langage de programmation C inclut le support d'un certain nombre d'opérateurs. Lister ces opérateurs, donner des exemples, leur associativité et comment elles sont appliquées.
5. Qu'est-ce qu'un opérateur ternaire? Donner sa syntaxe.
6. Dressez la liste des cinq types de données primaires et des quatre types de données secondaires
7. Quelle est la différence entre l'opérateur d'incrémentation postfixe et l'opérateur d'incrémentation préfixe?
8. Discutez les quatre types de constantes.
9. Décrivez trois façons de définir les constantes. Ecrire un programme langage C pour illustrer comment les constantes sont déclarées en utilisant les trois méthodes.

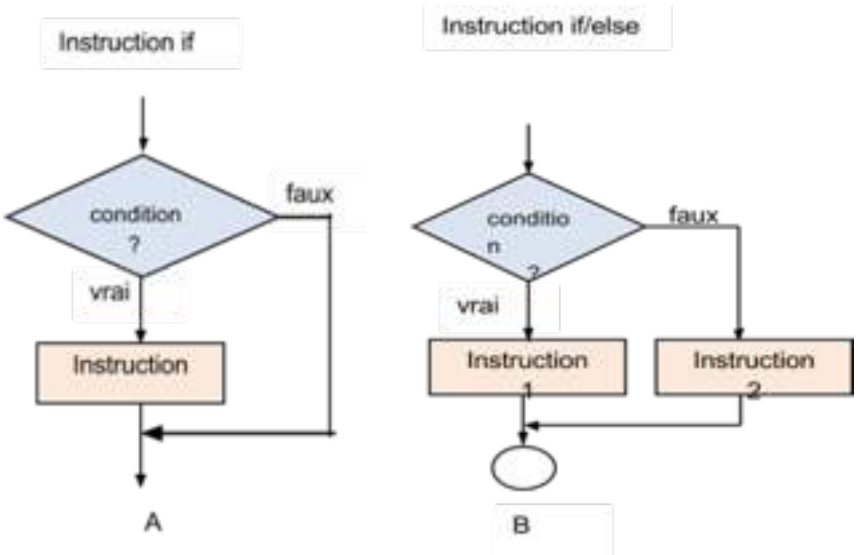
Activité 3.2 - Structures de contrôle et Fonctions

Introduction

Un programme ne se limite pas seulement à la circulation séquentielle de flux de contrôle, car à un certain moment, un programme peut décider d'utiliser un chemin particulier d'instructions ou peut répéter une instruction un certain nombre de fois tant que la condition est vrai. Par conséquent, d'autres styles de manipulation ou représentant les flux de contrôle dans la programmation incluent: structures de branchement et boucles. Les branchements sont également connus comme structures conditionnelles ou décisionnels. Les structures de boucle répètent une commande tant que la condition est remplie / vrai. Ces deux structures de contrôle font partie des composants de la programmation structurée (discuté dans l'unité 2) qui rendent un programme plus lisible et compréhensible.

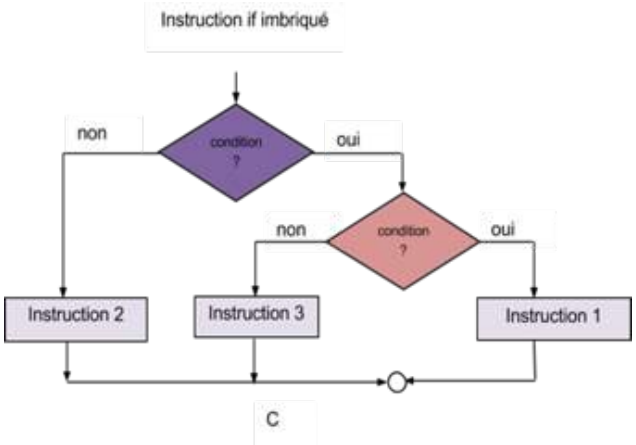
Structures de contrôle conditionnelles

Lorsqu'il s'agit de structures conditionnelles également connues sous le nom de instructions de sélection, le programmeur doit spécifier une condition qui doit être évalué par le programme, ainsi que les instructions à exécuter par le programme. Si la condition est évaluée et est Vrai, l'instruction if ou bloc d'instructions de la structure de sélection est exécuté (voir figure A). Une autre structure sélective est if/else. If/else évalue une condition et si elle est vrai, il exécute l'instruction if/else ou bloc d'instructions, mais si la condition évaluée est fausse, il exécute une autre instruction (voir la figure B).



Contrairement à la figure A, la figure B précise ce que nous voulons faire quand la condition est fausse. Dans la figure A, si la condition est fausse, le contrôle est passé à l'instruction suivant l'instruction if sans donner une autre option. Il existe également l'instruction if imbriquée où vous pouvez avoir une autre instruction if à l'intérieur (exécuté par une autre instruction if) d'une autre instruction if ou à l'intérieur d'une autre instruction if suivi par des instructions else if (voir figure C). Sa syntaxe est :

```
if (condition)    {
   if(condition) {
         instruction/s
   else
         instruction/s
}
else
instruction/s
}
```



Instruction Switch est une structure de contrôle de décisions multiples qui permet à plusieurs valeurs d'être testée par rapport à une liste de valeurs constantes jusqu'à ce qu'il trouve une correspondance et renvoie les résultats. C'est un moyen moins couteux de représenter le if/else imbriqué. C'est plus lisible et plus facile à comprendre que le if/else imbriqué.

```
switch(exp) {
        case 1:
                instruction/s;
                break;
        case 2:
                instruction/s;
                break;
                ......
......
        default:
                instruction;
                }
```

L'opérateur?: (Opérateur conditionnel) :

Cet opérateur fonctionne comme l'instruction if/else. Nous l'avons mentionné dans la première activité de cette unité. Sa synstaxe est comme ci-dessous:

exp 1? exp 2 : exp 3;

si exp 1 est vrai l'exp 2 est exécutée mais si exp 1 est fausse, alors exp 3 est exécutée.

Remarque:

L'instruction if ne se terminent pas avec (;) car c'est l'instruction qui la suit qui s'exécute. En outre, si l'instruction if exécute plus d'une instruction, il est important de les mettre dans un bloc de code en utilisant les accolades {}. Par exemple :

```
if (condition)
{
Instructions;
Instructions;
}
```

Structures de contrôle de répétition/boucle

Il y a des situations où une instruction doit être exécutée de façon répétée et une dans une on exécute une instruction le nombre de fois qu'on désire. Les instructions de boucles suivantes peuvent être utilisées pour exécuter une instruction à plusieurs reprises.

While: Cette déclaration répète une instruction un certain nombre de fois tant que la condition est vraie. Si la condition est fausse, On sort de la boucle While et les instructions qui la suivent sont exécutées (Les instructions qui ne sont pas dans les parenthèses de l'instruction while). Voir figure D. Sa syntaxe est:

```
while (condition)

{

instruction/s;

}
```

Par exemple:

```
#include <stdio.h>

int main () {

int x = 1;

while (x <= 10)

{

printf ("%d", x);

++x;

}

printf ("facile à utiliser!");


}
```

L'instruction for: Tout comme l'instruction while, il répète une boucle un certain nombre de fois. Voir la figure E. Sa syntaxe :

for(initialisateurr; condition; incrementation)

```
{
instruction/s;
}
```

Par exemple :

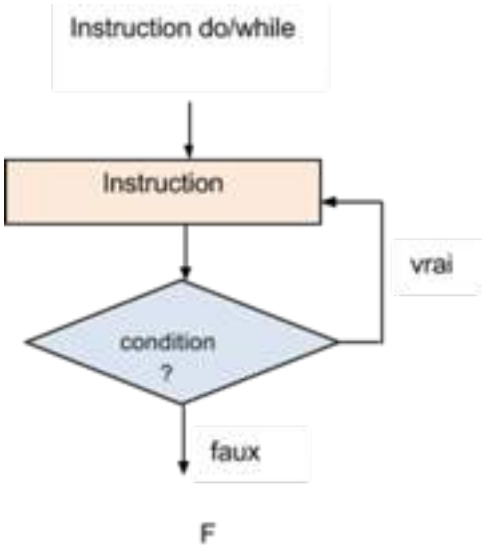
```
#include<stdio.h>
int main() {
for(int x=1; x<=10; ++x)
{
      printf("%d", x);
}
printf("Presque comme l'instruction while!");
}
```

L'instruction do/while: Voir la figure F. Sa syntaxe :

```
do {
instruction/s;
     }
```

Par exemple :

```
#include<stdio.h>
int main() {
int x=1;
do   {
   printf("%d", x);
++x;
} while( x<=10);
printf("easy to use!");
}
```



Remarque:

Il y a une petite différence entre while et do/while. L'instruction while teste d'abord la condition avant d'exécuter les instructions. Alors que le do / while exécute les instructions au moins une fois avant d'évaluer la condition - ce qui signifie qu'il renvoie une valeur au moins une fois quei la condition soit vraie ou fausse. Encore une fois, l'instruction do / while se termine par ; tout simplement parce qu'il n'exécute pas les instructions en dessous.

Instructions de saut

Il y a deux instructions de saut qui sont importantes dans la programmation. Ce sont: les instructions continue et break. Break arrête la boucle tandis que continue sort de la boucle et recommence à l'exécuter à l'itération suivante.

Fonctions

Dans l'unité précédente, nous avons abordé les techniques de résolution de problèmes qui incluent les approches top down et bottom up. Ces techniques consistent à diviser un programme en sous-programmes ou procédures et à les mettre ensemble plus tard pour former un programme. Vous en souvenez-vous? Sinon, relisez l'unité 2. Quoi qu'il en soit, ces techniques conduisent à une façon modulaire de programmer ou de créer des fonctions. Une fonction est un groupe d'instructions qui effectuent une tâche particulière. En termes simples, toutes les instructions dans un groupe spécifique, coopèrent pour atteindre un but.

La majorité des langages de programmation ont au moins une fonction appelée fonction main() et elle matérialise où commence l'exécution d'un programme. Les programmeurs sont également en mesure de réer ou d'ajouter d'autres fonctions à leurs programmes. Chaque fois qu'un programme rencontre un nom de fonction, le contrôle est ensuite donné à cette fonction particulière dont le nom a été rencontrée. Une fonction a cette forme:

```
Type-de-retour nom-fonction (liste-des-paramètres) // entête de la
fonction

{

instruction /s; // Corps de la fonction ou définition de la fonction

}
```

Type de retour : définit le type de données de la valeur que la fonction retourne. Certaines fonctions ne retournent aucune valeur après l'exécution des instructions. Ces fonctions sont définies avec le type de retour void.Comme un identifiant, un nom de fonction est le nom effectif de la fonction qui est utilisé par d'autres programmes chaque fois qu'ils ont besoin de ses services. Un paramètre est un espace réservé pour les valeurs qui sont données par l'environnement appelant. D'autres fonctions peuvent ne pas avoir de paramètres et la liste des paramètres peuvent être définis comme vide (void). Ce qui signifie que la fonction ne reçoit aucune valeur des autres programmes appelant. Le corps de la fonction contient des instructions qui définissent ce que fait la fonction.

Par exemple, une fonction qui renvoie Bonjour à une autre fonction :

```
void salutation (void)

{

printf ("Bonjour!");

}
```

Quand une autre fonction a besoin de la sortie ("Bonjour!") alors, le programmeur insère le nom de la fonction (salutation()) pour invoquer ou appeler cette fonction. La fonction qui appelle une autre fonction constitue l'environnement appelant. Maintenant, voyons le programme ci-dessous :

```
#include <stdio.h>

void salutation (void)

 {

         printf ("Bonjour!");

 }

main ()

{

salutations (); // Appel de la fonction

 }
```

Remarque:

Nous avons défini notre fonction (salutation()) avant notre fonction principale. C'est parce que le compilateur ne sera pas en mesure de la voir si nous l'avions placé après la fonction principale. Pour éviter de se limiter à l'endroit où une fonction est définir, on peut introduire le prototype de la fonction. Un prototype de fonction déclare une fonction et indique au compilateur qu'une fonction particulière existe dans le programme. Il donne les détails de cette fonction à savoir type de retour, nom et les paramètres. Par exemple :

```
#include <stdio.h>

void salutation(void); // Déclaration de fonction (prototype)

main ()

{

salutations (); // Appel d'une fonction

}

void salutation (void)

{

printf ("Bonjour!");

}
```

Portées des règles

Les variables peuvent être déclarées comme variables locales ou globales. Les variables locales sont définies à l'intérieur du bloc de code d'une fonction particulière. Ces variables ne peuvent être utilisées que par d'autres instructions qui sont à l'intérieur de cette fonction ou e bloc de code, simplement parce qu'ils ne sont pas visibles à l'extérieur de la fonction. Les variables globales sont définies en dehors de toutes les autres fonctions du programme. Par convention, ils sont déclarés au début du programme après les directives du préprocesseur mais avant la fonction main(). Elles gardent leurs valeurs pendant toute la durée l'exécution du programme et ils peuvent être accessibles par tout autre fonction définie dans le programme. Apprenons la portée des variables en utilisant cet exemple obtenu à partir site "point de tutoriel":

```
#include <stdio.h>

/* Déclaration variable globale */

int a = 20;

int main ()

{

/* Déclaration de variable locale dans la fonction principale
*/

int a = 10;
```

```
int b = 20;

int c = 0;

printf ("valeur de a dans la fonction main () = %d \n", a);

c = somme(a, b); // paramètres efectifs (a, b)

printf ("valeur de c dans main () =%d \n", c);

return 0;

}

/* Fonction pour additionner deux nombres entiers */

int somme (int a, int b) //Paramètres formels (int a, int b)

{

printf ("valeur de a dans somme() =%d \ n", a);

printf ("valeur de b dans somme() =%d \ n", b);

return  a + b;

}
```

Détails de l'activité

Tout comme la première activité de cette unité, vous devez lire plus sur les structures et les fonctions de contrôle. Cette activité est également pratique. Après lecture, vous devrez écrire des programmes simples qui vous aideront à comprendre les principaux concepts de la programmation. Il est conseillé d'utiliser les livres "Fundamentals of programming languages by Dipali P. Baviskar, Programming and PROB solving using C. by ISRD and Programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet". Vous pouvez également obtenir de pertinentes informations sur Internet. Par exemple sur le site http://www.tutorialspoint.com/cprogramming/index.htm

Après avoir lu sur ces sujets, cette activité vous demande :

1. Ecrie des notes brèves décrivant la différence entre structures de contrôle conditionnelles et structures répétitives.
2. Décrire l'importance des structures conditionnelles et des structures répétitives.
3. Selon votre propre compréhension, donner des exemples et éventuellement des illustrations, en expliquant les instructions de saut et leurs applications. Discuter de l'importance d'apprendre et de comprendre comment utiliser les instructions de saut.
4. Ecrire des notes brèves sur les avantages d'utiliser des fonctions dans un programme.

Conclusion

C'était une unité très intéressante où vous avez appris et implémenté quelques principes de base de la programmation. A présent, vous devez avoir cerné les concepts de programmation tels que les types de données, les variables, les opérateurs, les structures de contrôle et les fonctions. Ces concepts ont été abordés car ils sont communs à la plupart des langages de programmation.

Évaluation

1. Montrer la forme générale de :
   i. Instruction if
   ii. Instruction if/else
   iii. Instruction if imbriqué
   iv. Instruction if/else imbriqué
   v. Instruction conditionnelle
   vi. Instruction switch
   vii. Instruction while
   viii. Instruction do/while
   ix. Instruction for
   x. a function
2. Dans certaines situations, les structures de contrôle exécutent plus d'une instruction. Pourquoi est-il important d'utiliser les accolades {} lorsqu'il s'agit de plus d'une instruction?
3. Expliquer la diffrence entre do/while et while
4. Habituellement, les structures de contrôle par exemple if (x>y) ne terminent pas par un ; Pourquoi?
5. Quelle est la différence entre un paramètre formel et un paramètre effectif?
6. Qu'est-ce qu'un prototype de fonction? quelle est son importance dans un programme?
7. Quelle est la différence entre les variables globales et locales.
8. Citer quelques avantages d'utilisation de variables globales dans un programme?
9. Qu'est-ce que l'entête d'une fonction.
10. Donner les avantages d'utiliser des fonctions dans un programme.

Activité 3.3 – Les tableaux et les chaînes de caractères

Introduction

Un tableau correspond à une structure de données qui stocke des éléments du même type de données. Par exemple, pour stocker cinq entiers dans la mémoire, vous êtes tenus de déclarer cinq variables de type int (par exemple int a, b, c, d, e), mais un tableau est très pratique pour cette situation. En utilisant un tableau, vous créez un identifiant, puis définissez le nombre d'éléments ou la taille du tableau. Comme toutes les autres variables, un tableau doit être déclaré avant utilisation. Sa forme générale ressemble à ceci :

Type nom_tableau [taille];

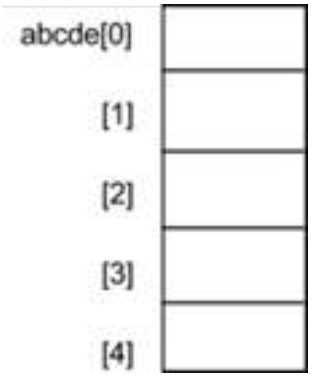
Le type indique le type des éléments qui seront stockés dans le tableau et inclut : int, float, char, double etc. Tandis que la taille indique le nombre maximum d'éléments qui peuvent être stockés à l'intérieur du tableau. Par exemple :

int abcde[5];

Float balance[10];

Les éléments du tableau sont accessibles en utilisant le nom du tableau et l'indice de l'élément. Les éléments de tableau sont stockés de manière contiguë et le premier élément du tableau commence avec un indice zéro (0), le dernier élément du tableau se termine avec un indice égal à la taille du tableau moins 1 ([taille-1]). Dans notre exemple, le premier élément est abcde [0] .... [4]. Voir le schéma:

Un tableau peut être initialisé comme:




```
Type nom_tableau []= {valeurs};

Par exemple;

abcde int [5] = {10, 20, 30, 40, 50};

ou;

int abcde [5];

abcde [0] = 10;
```

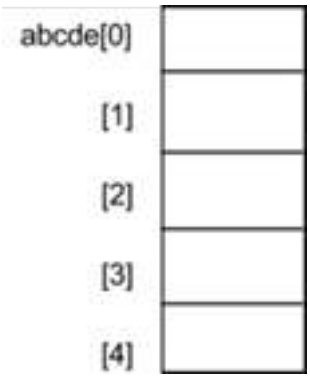
```
abcde [1] = 20;

abcde [2] = 30;

abcde [3] = 40;

abcde [4] = 50;

Voir le schéma :
```



L'instruction for est utilisée pour manipuler le contenu du tableau. Pour illustrer cela, prenons ce programme simple :

```
#include<stdio.h>

        main ()

{

            int abcde[5]={10, 20, 30,40,50}, x;

            for(x=0; x<5; ++x)  {

                printf("abcde[%d] =%d\n", x, abcde[x]);

                    }

        printf("J'ai affiché les éléments de mon tableau\n");

}
```

Sortie:

abcde [0] = 10;

abcde [1] = 20;

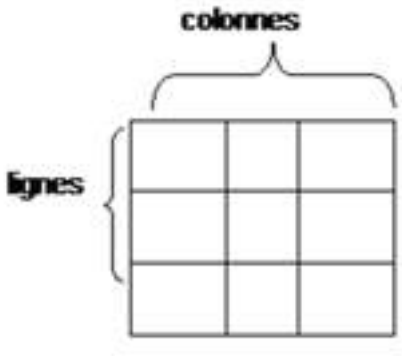
abcde [2] = 30;

abcde [3] = 40;

abcde [4] = 50;

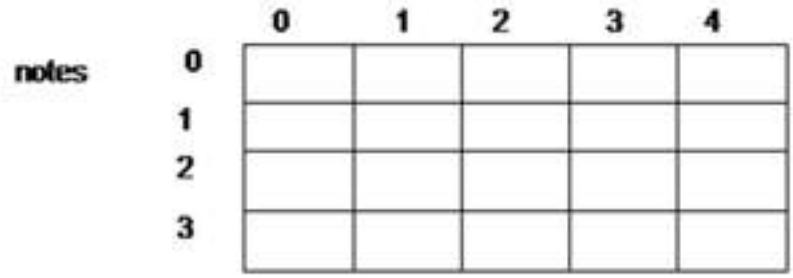
Ici, x est notre variable de contrôle que nous utilisons pour accéder ou manipuler le contenu du tableau.

Il existe d'autres types de tableaux connus appelés tableaux multidimensionnels. Un tableau multidimensionnel est un tableau de tableaux. Par exemple, un tableau à deux dimensions est un tableau multidimensionnel qui essaie de représenter des éléments sous forme de une table ou matrices. Il dispose de deux indices; une pour la ligne et l'autre pour la colonne. Par exemple, ci-dessous nous avons une table stockant neuf valeurs (3x3 = 9).



La forme générale d'un tableau à deux dimensions :

Type nom_tableau [taille];



Par exemple, ma_table est déclarée comme une matrice ayant 4 lignes et 5 colonnes. Le premier élément du tableau est ma_table [0] [0] et le dernier élément est ma_table [3] [4]. La figure ci-dessous illustre un tableau à deux dimensions;

Comme le tableau ci-dessus de dimension 1, ce tableau peut être initialisé comme suit:

Type [taille];

type nom_tableau [] = {valeurs}

Par exemple:

float notes[20] = {45.8, 34.2,56.0 ..... 78.5};

notes	0	1	2	3	4
0	45.8	34.2	56.0	66.7	76.8
1	78.5	54.3	49.5	52.4	36.0
2	55.5	54.5	44.3	66.5	80.5
3	90.5	77.5	48.0	56.6	78.5

Un programme simple pour illustrer les tableaux à deux dimensions

```
#include<stdio.h>

        main ()

{

float  notes[20] = {45.8,34.2,55.0,66.7,76.8,78.5,54.3,49.5,5
2.4,36.0,55.5,54.5,44.3,66.5,80.5,90.5,77.5,48.0,56.6,78.5},
x,y;

              for(x=0; x<5; ++x)

for(y=0; y<5; ++y)    {

                    printf("notes[%d][%d] =%d\n",
x,y,notes[x][y]);

                           }

printf("J'ai affiché les éléments de mon tableau à deux
dimensions\n");

}
```

Châines de caractères

Une chaîne de caratères est un tableau de caractères pouvant se terminer avec la valeur nulle '\ 0'. Ici, nous avons défini une chaîne de caratères en utilisant deux mots que nous avons rencontrées dans cette unité, à savoir tableau et caractère (char). Cela signifie qu'une chaîne de caratères est déclarée en utilisant le même format qu' un tableau:

Type nom_chaine [taille];

Type est char car c'est un tableau de caractères, nom_chaine désigne l'identifiant qui représente l'emplacement dans la mémoire pour stocker les caractères; taille désigne le nombre de caractères pouvant être contenus dans la chaîne.

Par exemple:

"Antilolas" est une chaîne entre guillemet double tandis que 'y' est un caractère enfermé dans un guillemet simple. Une chaîne de caractères prend la forme :

char prenom[10];

char nom[10];

Une chaîne de caractères peut être initialisée comme:

char prenom[10]= {'A, 'n', 't', 'i', 'l', 'o', 'l', 'a', 's','\0'};

ou :

char prenom[10]="Antilolas";

Par exemple:

```
#include <stdio.h>

int main ()

{

char prenom[10]= {'A, 'n', 't', 'i', 'l', 'o', 'l', 'a',
's','\0'};

char nom[10]="Rosianath";

printf("Salut! : %s %s\n", nom, prenom);

   return 0;

}
```

Détails de l'activité

Vous devez lire plus sur les tableaux et les chaînes de caractères. Après lecture, vous devrez écrire des programmes qui illustrent le concept de tableaux et de chaînes de caractères. Il est conseillé d'utiliser les livres "Fundamentals of programming languages by Dipali P. Baviskar, Programming and PROB solving using C. by ISRD and Programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet". Vous pouvez également obtenir de pertinentes informations sur Internet.

Par exemple http://www.tutorialspoint.com/cprogramming/index.htm . Cette activité vous oblige à faire une petite recherche. Après avoir lu ces sujets, cette activité vous demande de :

- Ecrire un programme qui trouve la somme des valeurs stockées dans un tableau à une dimension. Préciser la taille du tableau, le type de données et les valeurs.
- Ecrire un programme qui lit les valeurs d'un tableau à deux dimensions, puis trouve la moyenne des éléments. Donner le type du tableau, la taille et les valeurs. Afficher tous les éléments du tableau.
- Quelques fois, il est nécessaire de limiter l'utilisation multidimensionnelle des tableaux dans nos programmes. (effets des tableaux multidimensionnels sur notre système). Faire des recherches sur ce sujet.

Conclusion

Nous avons terminé cette unité en abordant deux concepts importants en matière de programmation: les tableaux et les chaînes de caractères. Comme vous pouvez le constater, ces deux notions sont presque incontournables lors de l'élaboration d'un programme ou lors de la conception d'une solution. Premièrement, nous traitons toujours des séries d'éléments de données, éventuellement du même type et deuxièmement on utilise des chaînes de caractères. Par conséquent, il est important que vous compreniez ces concepts. Enfin, la pratique vous aidera à maitriser la programmation.

Évaluation

1. Définir les termes suivants:
   i. un tableau
   ii. une chaîne de caractères
2. Pourquoi utilisons-nous l'instruction for imbriqué lorsqu'il s'agit d'un tableau à deux dimensions?
3. Quel problème se pose avec l'instruction suivante :

   Char nom [6] = {'J', 'e', 'a', 'n', 'n', 'e'} ; Expliquer votre réponse

3.4. Activité de laboratoire

Titre : Opérateurs, Structures de contrôle, tableaux et organigramme

Objectifs du lab

L'objectif du lab est d'emmener l'étudiant à réfléchir sur comment résoudre un problème. Il doit pourvoir à l'issu de ce lab capter la logique de résolution d'un problème donné. L'autre objectif est de l'emmener à utiliser de façon adéquate les structures de controle. Le lab met également l'accent sur l'évaluation des expressions en tenant compte de l'ordre de priorité des opérateurs. L'étudiant apprendra également à faire des organigramme à partir un programme.

Exercice 1

Objectif de l'exercise : Apprendre à évaluer les expressions

Resources à utiliser :

Temps nécessaire : 1h

Description de l'exercice

a. Soient les déclarations suivantes :

   int n = 5, p = 9 ;

   int q ;

   float x ;

   Quelle est la valeur affectée à la variable q ou à la variable x dans chacune des instructions suivantes ?

   1- q = n < p ;

   2- q = n == p ;

3- q = p % n + p > n ;

4- x = p / n ;

5- x = (float) p / n ;

6- x = (p + 0.5) / n ;

7- x = (int) (p + 0.5) / n ;

8- q = n * (p > n ? n : p) ;

9- q = n * (p < n ? n : p) ;

b. Enlever les parenthèses des expressions suivantes lorsqu'elles peuvent être retirées.

1- a=(25*12)+b;

2- ((a>4) &&(b==18))

3- ((a>=6)&&(b<18))||(c!=18)

4- c=(a=(b+10));

Évaluer ces expressions pour a=6, b=18 et c=24. On supposera que les valeurs données le sont pour chacune des lignes : il n'y a pas d'exécution séquentielle comme dans un programme.

Instruction pour la soumission : Soumettre les réponses et mettre les codes dans un document Word

Critère d'évaluation : pour a) et b) donner 1 point pour chaque bonne réponse

Liens ou références :

- http://thoorens.net/notes/pmp
- http://www.ltam.lu/cours-c/prg-c_c.htm

Exercice 2

Objectif de l'exercise : Apprendre à manipuler les structure de contrôle et à dessiner les organigrammes

Resources à utiliser : Ordinateur, Environnement de Développement Intégré (Dev C++ par exemple)

Temps nécessaire : 3h

Description de l'exercice

En utilisant un logiciel (de préférence Microsoft Word), dessinez un organigramme pour effectuer les étapes indiquées dans les exercices 1 à 9 puis donnez le programme C correspondant. Supposons que les déclarations sont correctement faites.

1. Lire un entier, représentant l'âge. Si l'âge est égal à 20, incrémenter l'âge de 1. Afficher "l'année prochaine vous aurez ----- ans" (Afficher le contenu de la variable)
2. Si la note est inférieure à 10, afficher, "Votre travail est insuffisant". Sinon si la note est comprise entre 10 et 11, afficher "vous avez la mention «passable»", sinon si la note est comprise entre 12 et 14, afficher, "vous avez la mention «assez-bien»", sinon si la note est comprise entre 15 et 16, afficher, "vous avez la mention «bien»", sinon afficher, vous avez "la mention «très bien»".
3. Si la taille est supérieure ou égale à 53,9 mètres, alors décrémenter le poids de 20,4 kg. Sinon afficher "maintenir le poids à 55 kg".
4. Tant que x est inférieur ou égal à 15, afficher la somme des valeurs entier allant de 1 à x.
5. Traduire les instructions du 4 avec l'instruction for
6. Traduire les instructions du 4 avec l'instruction do/while
7. Ecrire des programmes simples implémentant les exercices (organigrammes 1-7), Vous devez déclarer les variables.

Ecrire également un programme qui lit entier n (n <10), le passe à une fonction qui fait n itérations en affichant un message de votre choix.

Instruction pour la soumission : Soumettre les réponses et mettre les codes dans un document Word

Critère d'évaluation : 2 points par question

Liens ou références :

- http://thoorens.net/notes/pmp
- http://www.ltam.lu/cours-c/prg-c_c.htm

Exercice 3

Objectif de l'exercice : Développer la logique de résolution de problème chez l'apprenant

Ressources à utiliser : Ordinateur, Environnement de Développement Intégré (Dev C++ par exemple)

Temps nécessaire : 2h

Description de l'exercice

a. Ecrire un programme qui demande à l'utilisateur de saisir une série de nombres réels non nuls. Pour arrêter la saisie, l'utilisateur entre la valeur zéro. Le programme affiche la moyenne des valeurs saisie.

b. Ecrire un programme qui demande à l'utilisateur d'entrer un nombre entier n et affiche les diviseurs de ce nombre qui sont des nombres premiers. S'il n'y a pas de diviseurs nombres premiers, on informe l'utilisateur.

Instruction pour la soumission : Soumettre les réponses et mettre les codes dans un document Word

Critère d'évaluation : a) 5 points b) 8 points

Liens ou références :

- http://thoorens.net/notes/pmp
- http://www.ltam.lu/cours-c/prg-c_c.htm

Exercice 4

Objectif : Emmener l'étudiant à maitriser la manipulation simple des tableaux.

Ressources à utiliser : Ordinateur, Environnement de Développement Intégré (Dev C++ par exemple)

Temps : 1h

Description de l'exercice

a. Écrire un programme qui lit deux tableaux A et B et leurs dimensions N et M au clavier et qui ajoute les éléments de B à la fin de A.

b. Écrire un programme qui demande à l'utilisateur de saisir la dimension N d'un tableau T du type int de dimension maximale 50.

- Remplir le tableau par des valeurs entrées au clavier et afficher le tableau.
- Calculer et afficher ensuite la somme des éléments du tableau.
- Copiez ensuite toutes les composantes strictement positives dans un deuxième tableau A et toutes les valeurs strictement négatives dans un troisième tableau B.
- Afficher le contenu des tableaux A et B.
- Trouver le plus petit élément et le plus grand élément dans chacun des tableaux A et B et afficher leur contenu.

Instruction pour la soumission : Soumettre les réponses et mettre les codes dans un document Word

Critère d'évaluation : a) 5 points b) Attribuer 2 points à chaque ligne de question

Liens ou références :

- http://thoorens.net/notes/pmp

Résumé de l'unité

Cette unité a mis l'accent sur la théorie et la pratique des principes de la programmation. Il a été présenté à l'étudiant les concepts de résolution de problèmes en utilisant le langage C. Par conséquent, l'unité a présenté les environnements de développement en C de même que des programmes simples et l'utilisation des fonctions printf() et scanf(). Elle a également examiné les types de données, les variables, les constantes, les opérateurs, les structures de contrôle, les fonctions, les tableaux et les chaînes de caractères. A présent, l'étudiant doit être capable d'écrire des programmes en utilisant toutes les constructions qui ont été enseignées dans cette unité.

Évaluation de l'unité

Vérifiez votre compréhension!

Questions

1. Déclarer et initialiser une variable d'entier appelé total à 0 (2 points)
2. Déclarer et initialiser un caractère appelé initial à K (2 points)
3. Utiliser le mot-clé const pour déclarer une constante PI et attribuer lui la valeur 3,14 (3 points)
4. Définir une constante PI qui a la valeur 3,14 (2 points)
5. Affectez la valeur de la variable de type entier a à la variable b (2 points)
6. Stocker la somme de deux variables c et e dans la variable sum (3 points)
7. Ajouter la valeur de a à la valeur de b et stocker la réponse dans a (2 points)
8. Divisez la valeur de la variable sum par 9 et stocker la réponse dans sum (3 points)
9. Lire un nombre entier dans y (2 points)
10. Lire un caractère dans c (2 points)
11. Affectez le résultat de la division de la variable entier total par 5 dans la variable balances de type float. Utiliser le transtypage pour s'assurer que le reste est également stocké par la variable de type float. (3 points)
12. Écrire une boucle while pour afficher toutes les valeurs de 1 à 15 à l'écran (3 points)
13. Écrire une boucle for afficher les valeurs 3 à 7 (3 points)
14. Écrire une boucle for qui additionne toutes les valeurs entre 50 et 100 dans une variable appelée sum. (3 points)

15. Ecrire une instruction if qui compare la valeur d'un entier appelé produit à la valeur 15, et si elle est supérieure, affiche la chaîne de caractères "vous avez dépassé". (4 points)

16. Si la variable x est égal à y, afficher la valeur de x, sinon afficher la valeur de y. (4 points)

17. Réécrire le code suivant en utilisant l'instruction switch (4 points)

```
if( c == 'A' )
      sum = 0;
else if ( c == 'B' )
      sum+= 1;
else if( c == 'Z' )
      sum = 26;
else
      printf("Caractère inconnu %c\n", c );
```

18. Convertir l'expression suivante avec une instruction if / else v = (a <b)? a: b; (2 points)

19. Déclarer un tableau unidimensionnel appelé limites qui stocke six entiers (2 points)

20. Attribuer 23,46,67,87,53,44 aux éléments du tableau limites (2 points)

21. Affecter 56 à l'élément 4 du tableau limites (2 points)

22. Affecter 34 au cinquième élément du tableau limites (2 points)

23. Afficher la valeur du premier élément du tableau limites (2 points)

24. Déclarer un tableau de caractères de 20 éléments appelé noms (3 points)

25. Affecter le caractère «S» à l'élément 10 du tableau noms. (4 points)

26. Écrire une boucle for pour lire les valeurs du tableau limites (19). (4 points)

27. Écrire une boucle for pour lire les caractères du tableau noms (24) . (4 points)

Instructions

Vous devez écrire des déclarations simples pour toutes les questions 1 à 27.

Les questions couvrent tout ce qui a été présenté dans cette unité afin d'évaluer la compréhension globale. Répondez attentivement et si votre score tombe

- en dessous de 40%, refaire les lectures et activités.
- entre 40% et 60%, refaire les lectures sur vos points faibles
- supérieur à 60%, vous avez un bon niveau de connaissances

Système de notation

Devoirs 20%

Evaluation 10%

Examen final 70%

Réponses

1. int total=0;
2. char initial='K';
3. float const PI = 3.14;
4. #define PI 3.14
5. int a, b;

   b=a;
6. sum=c+e;
7. a+=b; or a=a+b;
8. sum/=9;
9. scanf("%d", &y);
10. scanf("%c", &c);
11. balances=(float) total/5;
12.
```
int values=1;
while (values<=15) {
printf("%d\n", values);
++values ;
 }
```

13. for(int values=3; values<=7; ++values)

14. for(int values=51, sum=0; values<100; sum+=values, ++values) or

```
int sum=0, values;
for(values=51; values<100; ++values)
sum+=values;
```

15. if(produit>15)

```
printf("vous avez dépassé");
```

16.
```
if(x==y)
printf("%d", x);
else
printf("%d", y);
```

17.
```
 switch (c) {
case 'A':
         sum = 0;
         break;
        case 'B':
         sum+= 1;
         break;
        case 'C':
         sum = 26;
         break;
     default:
               printf("Caractère inconnu %c\n", c);
   }
```

18. if (a<b)

```
  v=a;
else
    v=b;
```

19. int limites [6];

20. limites [6]= {23,46,67,87,53,44};

```
21.  limites[4]= 56;
22.  limites[4]=34;
23.  printf("%d",limites[0]);
24.  char noms[20];
25.  noms[10]='S';
26.  for (x=0; x<6; ++x)
       scanf("%d" &limites[x]);
27.  for (x=0; x<20; ++x)
       scanf("%s", noms);
```

Lectures unitaires et d'autres ressources

- "Achille Braquelaire", Méthodologie de la programmation en C, Norme C 99 - API POSIX, Dunod - 672 pages, 4eédition, 1er mars 2005
- "Yves Mettier",, C en action Solutions et exemples pour les programmeurs en C,
- Fundamentals of programming languages. Dipali P. Baviskar, Technical Publication Pune.
- Programming and problem solving through "C" languages. Harsha Priya, r. Ranjeet. Firewall media, 2006.
- Programming and PROB solving using C. ISRD, Tata McGraw-Hilleducation.
- Lectures et autres ressources optionnelles:
- Fundamentals of Programming: with Object Oriented Programming, Gary Marrel, 2009, Gary Marrer.
- Programming Logic and design, Comprehensive, Joyce Farrel, 2012, 7th edition. Cengage Learning.
- Programming: Grade in Industrial Technology Engineering. Creative Commons. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf

Unité 4.Test de programme, Débogage et Documentation

Introduction à l'unité

Cette unité porte sur les différents types d'erreurs qui surviennent au cours du développement d'un programme et des diverses façons de corriger. Elle s'intéresse également sommairement à la documentation lors du développement logiciel. Cette unité permettra à l'apprenant de se familiariser aux trois concepts essentiels (tests, débogage et documentation) qui sont communs à tous les langages et paradigmes de programmation. Le contenu de cette unité est basé sur les livres en référence. Cependant, vous êtes invités à faire recours à ces ressources pour plus de lecture afin d'être en mesure de travailler convenablement sur les taches/activités.

Objectifs de l'unité

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

- Définir les concepts de tests et de débogage d'un programme.
- Appliquer les types de base de tests d'un programme.
- Différencier les types de base de tests d'un programme.
- Déboguer un programme.
- Documenter un programme à l'aide des commentaires et expliquer d'autres types de documentation d'un logiciel.

Termes clés

Test: C'est un processus d'exécution d'un programme dans le but de trouver une erreur.

Débogage: C'est un processus méthodologique afin de trouver et de réduire le nombre de bugs ou de défauts dans un programme informatique permettant ainsi au programme de se comporter comme prévu.

Documentation: Elle comporte des instructions pour mieux utiliser un programme

Débogueurs: Ce sont des programmes qui permettent d'exécuter un programme de manière contrôlée, afin d'être en mesure d'examiner le contenu du programme pour y détecter un bug logique ou d'exécution.

Plan de test: un plan de test de logiciels est un document décrivant la portée et les activités de test. C'est la base pour tester formellement tout logiciel / produit dans un projet. Il décrit comment on pense tester un programme, quelles sont les entrées qui doivent être testées et pourquoi elles ont été choisies.

Un bug: Un bug logiciel est un problème qui fait qu'un programme se plante ou produit des résultats incorrects. Le problème est causé par une logique insuffisante ou erronée. Un bug peut être une erreur, une faute, une anomalie ou un défaut, qui peut entraîner une défaillance ou une discordance des résultats attendus. La plupart des bugs sont dus aux erreurs humaines dans le code source ou dans sa conception. Un programme est dit « buggy » quand il contient un grand nombre de bugs qui affectent les fonctionnalités du programme et provoquent des résultats incorrects.

Activités d'apprentissage

Activité 4.1 – Erreurs de programme et test

Introduction

Il existe trois types d'erreurs de base à savoir : syntaxe ou erreurs de compilation, erreurs d'exception à l'exécution et erreurs logiques. Un programme doit être testé pour tous ces types d'erreurs. Cela garantit la fiabilité, l'efficacité, la performance, entre autres attributs d'un bon programme. Le test est d'abord divisé en deux types de base à savoir : tests boîte blanche (white box testing) et tests boîte noire (black box testing). En utilisant la méthode de la boîte blanche, un programmeur peut tester des cas qui garantissent que chaque partie individuelle dans un programme a été exécutée au moins une fois, toutes les décisions logiques sur leur valeurs vraies et fausses sont exécutées, toutes les boucles dans leurs limites opérationnelles sont exécutées et les structures de données internes pour s'assurer de leur validité sont exécutées.

Le test boîte noire tente de trouver des erreurs dans les catégories suivantes : 1. fonctions incorrecte ou manquante, 2. erreurs d'interface, 3. erreurs dans la structure de données ou l'accès à une base de données externe, 4. erreurs de performance, 5. initialisation et erreurs de terminaison.

Le test boîte noire est une approche complémentaire qui est susceptible de couvrir les autres types d'erreurs que le test boîte blanche ne prend pas en compte. Il est pratiquement impossible de prouver que le programme est correct sur toutes les entrées. En revanche, les sceptiques doivent être convaincus que cela fonctionne la plupart du temps juste en testant le programme sur les différentes entrées et en documentant ce qui a été fait. Ce document est appelé plan de test et doit être fourni pour chaque programme.

Détails de l'activité

Par conséquent, cette activité consiste à lire et écrire un programme simple qui sera utilisé pour maitriser les concepts de tests boîte blanche et boîte noire. Vous devez lire les sujets sur les types d'erreurs qui apparaissent au cours des techniques de développement et sur les techniques de tests de programme qui incluent les tests boite blanche et les tests boite noire. Vous pouvez lire : Programming and problem solving through 'C' languages par Harsha Priya, r. Ranjeet et Fundamentals of programming languages par Dipali P. Baviskar. Ces ouvrages de référence sont répertoriés dans la section référence pour cette unités. Vous pouvez également utiliser tout autre matériel pertinent trouvé sur Internet. Pour accomplir la tâche au point 3, de cette activité, l'apprenant est tenu de consulter les matériels de lecture fournis dans la section de référence de l'unité 3 sur les fonctions.

Dans cette activité, vous vous devez :

- Faire la différence entre syntaxe, erreurs logiques et erreurs en temps exécution.
- Décrire comment les tests boite blanche et boite noire fonctionnent.
- En utilisant le langage C, écrire un programme simple qui demande à l'utilisateur une valeur (lue dans un entier n), qu'il passe à une fonction qui renvoie la somme des valeurs de 1 à n.
- Tester le programme ci-dessus en utilisant le test boite blanche et le test boite noire. Décrire comment vous avez réussi à tester votre programme en utilisant les deux techniques. Pour le test boîte noire, vous êtes tenu d'utiliser différents types de données. Comment le programme répond t-il à ce changement?
- D'une manière très simple, décrire le cas de test que vous avez utilisé pour accomplir la tâche ci-dessus.

Conclusion

A ce stade de l'étude, l'apprenant doit être bien équipé avec les connaissances sur les types d'erreurs de programmation et techniques de test d'erreur qui incluent les tests boite blanche et boite noire.

Évaluation

1. Nommer et décrire brièvement trois types d'erreurs d'un programme. Donner un exemple pour chaque type d'erreur.
2. Pourquoi est-il nécessaire de tester un programme même si nous savons qu'il fonctionne bien?
3. Présenter deux types de tests de programme.
4. Donner une façon simple quelconque d'effectuer le test boîte blanche.

Activité 4.2 – Débogage de programme

Introduction

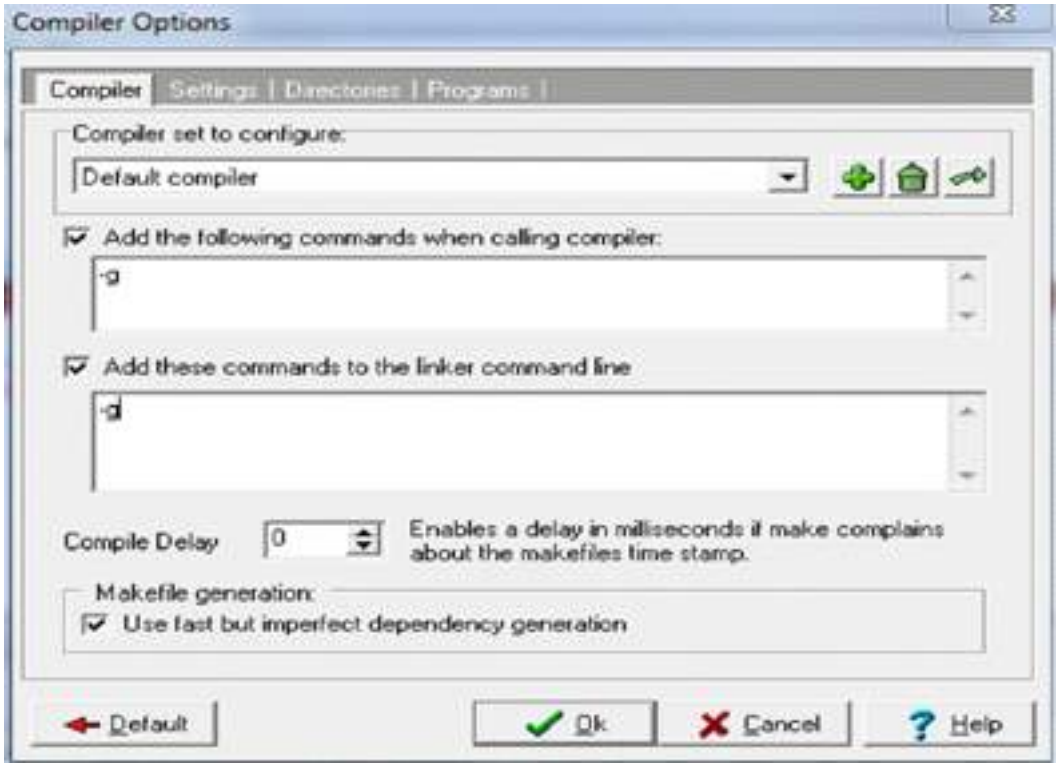
Le débogage implique un processus de recherche et de réduction du nombre d'erreurs en s'assurant que le programme fait exactement ce qu'il est censé faire. Ce processus a tendance à être plus difficile lorsque plusieurs sous-systèmes / sous-programmes / procédures dépendent fortement les uns des autres ou sont étroitement couplés, de telle sorte que les changements dans l'un des sous-programmes peuvent avoir un impact sur d'autres sous-programmes dépendants faisant ainsi émerger des erreurs dans d'autres sous-programmes.

Test	Débogage
Le test trouve les cas où un programme ne répond pas à la spécification	Le processus de débogage consiste à analyser et éventuellement étendre un programme qui ne répond pas aux spécifications afin de trouver un nouveau programme proche de l'original qui ne répond pas aux spécifications.
Son objectif est de démontrer que le programme répond aux spécifications de conception	L'objectif est de découvrir la cause exacte et supprimer les erreurs détectées dans le programme.
Le test est terminé lorsque toutes les vérifications souhaitées par rapport aux spécifications ont été réalisées	Le débogage est terminé lorsque toutes les erreurs connues dans le programme ont été corrigées

Le test peut commencer dans les premiers stades de développement de logiciels	Le débogage ne peut commencer que lorsque le programme est codé
Le test est le processus de validation de l'exactitude du programme	Le débogage est le processus d'élimination des erreurs dans un programme

Les erreurs logiques peuvent être détectées en faisant la simulation manuelle du code du programme et en mettant les instructions d'impression dans le code de programme. Une autre approche pour détecter les erreurs logiques est par l'utilisation d'un débogueur. C'est la technique la plus couramment utilisée. Le débogueur est le programme qui aide un programmeur à suivre l'exécution étape par étape de son programme en permettant l'affichage des résultats de calculs intermédiaires du programme.

L'IDE comprend un débogueur, qui est un outil de support pour trouver des erreurs d'exécution. Pour utiliser le débogueur, il est nécessaire de modifier les options du compilateur afin d'inclure des informations de débogage dans les fichiers objet et exécutables (Outils> Options> Compiler compilateur). Cela se fait en ajoutant le paramètre -g pour les appels vers le compilateur et l'éditeur de liens.



source(http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf)

Lorsque le programme est compilé et l'édition de lien fait avec ces options, nous pouvons exécuter le programme en mode débogage en cliquant sur le bouton de débogage (F8). Ce mode permet de lancer l'étape d'exécution du programme pas à pas (instructions sont exécutées une par une), de fixer l'arrêt de l'exécution à des points d'arrêt, ou de regarder la valeur d'une variable à tout moment de l'exécution.

Le mode Debug nous permet de consulter ou regardons la valeur d'une variable à tout moment pendant l'exécution du programme. Pour regarder une variable, le programme doit être arrêté (avec un point d'arrêt ou après l'utilisation de "Run to Cursor").

Par exemple, lorsque le code suivant est compilé, puis exécuté dans le programme C: Obtenu à partir de : (http://faculty.kfupm.edu.sa/ICS/said/ics103_101/Lab03%20Supplement%20-%20How%20to%20use%20Dev%20C++%20Debugger.doc).

```
#include <stdio.h>

#include <stdlib.h>

int main(void)

{

    int x = 5;

    double a, b;

    printf("Enter the first number (double type): ");

    scanf("%lf", &a);

    printf("\n a = %f", a);

    a = a + x;

    printf("\n a = %f", a);

    printf("\n Enter the second number (double type): ");

    scanf("%f", &b);

    printf("%.2f  +  %.2f  =  %.2f\n", a, b, a + b);

    printf("x = %d\n", x);

 system("PAUSE");

    return 0;

}
```

La sortie suivante est obtenue pour les valeurs d'entrée a = 5,5 et b = 7,5:

La sortie indique que le programme a une ou plusieurs erreurs logiques. Le débogueur de Dev C++ peut être utilisé pour trouver l'erreur (s).

Détails de l'activité

Pour exécuter cette activité, l'apprenant est invité à lire sur le débogage de programme. C'est-à-dire comment déboguer syntaxe et erreurs logiques en utilisant le livre « fondamentaux des langages de programmation » de P. Dipali Baviskar et le lien http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf pour apprendre comment utiliser un débogueur; créer des points d'arrêt et des variables à visualiser (watches). Ces matériaux vous aideront à comprendre le programme de débogage et vous guideront étape par étape pour apprendre à déboguer un programme. Dans cette activité, vous devez:

- Élaborer un programme C qui demande à l'utilisateur deux valeurs entières, calcule le modulo, et affiche le résultat à l'écran. Ajouter des points d'arrêt et des watches à toutes les variables du programme. Exécutez le programme pas à pas et vérifier la façon dont les valeurs sont modifiées par les instructions. Quelle est la valeur des variables avant de les lire à partir du clavier?
- Décrire comment vous avez accompli la tâche ci-dessus.
- Utiliser un programme C, créer un nouveau projet en Dev C ++. Ajouter des points d'arrêt et des watches.

Conclusion

Cette activité a été passionnante et jusqu'à présent nous avons appris comment déboguer les erreurs de programmation en faisant la simulation manuelle du code de programme et en mettant les instructions d'impression dans le programme et en outre, par l'utilisation d'un débogueur qui est la technique la plus couramment utilisée. Le débogueur est un programme logiciel qui aide un programmeur à suivre l'exécution étape par étape de son programme en permettant l'affichage des résultats de calculs intermédiaires du programme. Nous avons appris à ajouter des points d'arrêt et watches à notre code avec l'utilisation d'un débogueur.

Évaluation

1. Qu'entendez-vous par le terme bug?
2. Dressez la liste des différences entre tests et débogage?
3. Décrivez comment déboguer syntaxe et erreurs logiques

Activité 4.3 – Documentation de programme/logiciel

Introduction

La documentation est un aspect très important de la programmation, parce que dans votre carrière de programmeur, vous pourriez lire beaucoup plus de code, que vous en écriviez. Si la documentation d'un programme est appropriée, alors comprendre le programme est facile car en lisant le programmeur écrit le programme2 - Harsha Priya, r. Ranjeet. La documentation est souvent catégorisée en : installation, référence et tutoriel. La documentation présente de nombreux avantages qui comprennent : utilisation facile d'un programme, maintenance facile, modification facile, compréhension facile de la logique d'un programme à partir des enregistrements documentés plutôt qu'à partir du code même, organigrammes ou commentaires utilisées dans le programme, très utiles pour la compréhension de la fonctionnalité de l'ensemble du programme, les enregistrements documentés sont très utiles pour le redémarrage d'un projet de logiciel qui a été reportée pour une raison ou l'autre, et probablement, le travail ne doit pas être recommencé à partir de zéro et les vieilles idées peuvent encore être facilement récapitulé ; ce qui fait économiser beaucoup de temps et évite la duplication des travaux (réutilisabilité). Diverses formes de documentation incluent : commentaires, manuel système et mode d'emploi.

Détails de l'activité

C'est aussi une activité de lecture et l'apprenant est invité à lire sur la documentation programme/logiciel utilisant "Fundamentals of programming languages book by Dipali P. Baviskar ". Nous allons pratiquer le concept de la documentation d'une manière simple par l'utilisation des commentaires. L'apprenant peut visiter http://en.wikibooks.org/wiki/C_Programming/Structure_and_style pour lire sur les styles de documentation par l'utilisation des commentaires. Dans cette activité, vous devez :

- Décrire brièvement un code d'auto-documentation. Utiliser un exemple simple
- Décrire manuels utilisateur et système
- En utilisant des commentaires, documenter le programme suivant :

```
#include<stdio.h>

void sms_me(int i)

{

for(; i>0; --i)

printf("Je suis un programmeur!\n");

}

main ()
```

```
{

int x;

print("Entrer un entier qui n'est pas grand\n");

sms_me(x);

printf("Ainsi, tu vas programmer les activités de mon
département\n");

}
```

- Utiliser le langage C, pour proposer un programme de ton choix. Documente le en utilisant à la fois les styles de commentaires d'une ligne et de plusieurs lignes. Dans votre programme, montrer le concept de l'auto documentation d'un programme.

Conclusion

Cette activité présenté la documentation de logiciel, une activité qui est essentielle et mise en œuvre les programmeurs. Le point important à ce niveau est d'aider l'étudiant à comprendre le sens de l'auto documentation de code et la façon de documenter le code en utilisant commentaires. L'étudiant a également appris les manuels d'utilisation et systèmes dans le cadre de la documentation de logiciel. A ce niveau, l'apprenant doit être en mesure d'apprécier les avantages de la documentation de logiciel.

Évaluation

1. Qu'entendez-vous par le terme documentation logiciel/programme?
2. Discuter les avantages de la documentation de programme
3. Discuter les trois catégories de la documentation de logiciel
4. Expliquer brièvement les formes de documentation de programme / logiciel

Résumé de l'unité

Cette unité a étudié les trois principaux concepts de programmation qui sont importants et qu'un nouvel étudiant en programmation a besoin de connaitre. Ces concepts sont : le test d'un programme, le débogage et la documentation.

Évaluation de l'unité

Questions

1. Définissez les paramètres suivants (7 points)

   i. Un bug_____

   ii. Test de programme _____

   iii. Débogage_____

2. Les erreurs de compilation sont détectés par_____ (1 point)

3. Les erreurs d'exécution sont également connues comme_____ (1 point)

4. Une instruction de programme a été écrite : (2 points)

   x = d \ y; quelle erreur est générée?

5. Faire correspondre : (4 points)

a.Tests boîte blanche	i. cas de test qui garantissent que chaque partie individuelle dans un programme a été exécutée au moins une fois, toutes les décisions logiques sur leurs valeurs vraies et fausses sont exécutées, toutes les boucles dans leurs limites opérationnelles sont exécutées et les structures de données internes pour s'assurer de leur validité sont exécutées.
b Tests boite noire	ii.un ensemble de conditions ou variables dans lesquelles un testeur détermine si un système en cours de test satisfait aux exigences ou fonctionne correctement.
c. Plan de test	iii. document décrivant le but des tests et les activités
d. cas de test	iv. Essaient de trouver des erreurs comme les fonctions incorrectes ou manquantes, les erreurs d'interface, les erreurs dans les structures de données ou l'accès aux bases de données externes, les erreurs de performance, et les erreurs d'initialisation et de terminaison par la saisie de données.

6. Pour regarder une variable, le programme doit être arrêté avec un _____ (2 points)

7. Les erreurs logiques peuvent être détectées par _____ et _____ (2 points)

8. Les types de documentation peuvent être regroupés en trois catégories. Nommez les trois catégories de la documentation de logiciel. (3 points)

Directives

Les questions couvrent tout ce qui a été présenté dans cette unité afin d'évaluer la compréhension globale. Répondez attentivement et si votre score tombe :

- en dessous de 40%, refaire les lectures et activités.
- entre 40% et 60%, refaire les lectures sur vos points faibles
- supérieur à 60%, vous avez un bon niveau de connaissances

Système de notation

Devoirs 20%

test d'évaluation 10%

Examen final 70%

Réponses

1. i. Un bug est un problème qui amène un programme à planter ou à fournir un résultat invalide. Le problème est causé par une logique insuffisante ou erronée. Un bug peut être une erreur, un défaut ou une faute, qui peut entraîner une défaillance ou une déviation des résultats attendus.

   ii. Le test est un procédé d'exécution d'un programme dans le but de trouver une erreur.

   iii. Le débogage est un processus méthodologique pour trouver et réduire le nombre de bugs ou de défauts dans un programme informatique lui permettant de se comporter comme prévu.

2. le compilateur
3. erreurs d'exécution
4. syntaxe
5. a i
   b iv
   c iii
   d. ii
6. point d'arrêt

7. simulations manuelle et l'aide d'un débogueur
8. installation, référence et tutoriel

Résumé du module

Dans ce module, vous avez appris les bases de la programmation. Dans l'unité 1, vous avez appris les générations des langages de programmation et les paradigmes de programmation. Par la suite, l'unité 2 a présenté les différentes techniques de résolution de problèmes à un bas niveau, y compris l'identification, l'analyse, la conception de la solution. Elle aborde aussi la traduction des solutions trouvées sous forme d'organigrammes et / ou de pseudo-codes. L'unité 3 quant à elle, vous a permis d'étudier les notions de bases de la programmation. Ces notions vous ont permis d'écrire de simples programmes en langage C. Enfin, l'unité 4 énumère les différents types d'erreurs qui apparaissent au cours du développement d'un programme et comment les déboguer. Elle met aussi l'accent sur l'importance de la documentation en matière de développement de logiciels.

Evaluation du cours

Cette section consiste à évaluer le cours dans sa globalité. Elle propose donc 3 Examens.

Examen I : Descriptive

Instructions

Répondez à toutes les questions en 3 heures, puis consulter le document de référence ou vos notes pour cette unité. Vérifiez votre compréhension en marquant ce que vous avez écrit.

Système de notation

Question 1 : 0.5*7 points

Question 2 : 0.5*8 points

Question 3 to 11 : 1 point par question

Question 12 et 13 : 1 point par question

Question 14 : 4 points

Question 15 et 16 : 1 point par question

Question 17 : 2.5 points

Question 18 : 1 point

Question 19 : 2 points

Question 20 : 4 points

Enoncé :

1. Définissez les termes suivants :

   i. Programmation

   ii. Syntaxe

   iii. Sémantique

   iv. Code de programme

   v. Algorithme

   vi. Organigramme

   vii. Bug

2. À l'aide d'exemples, différencier entre :

   i. Opérateurs unaires et binaires

   ii. Compiler et traducteur

   iii. Langages bas niveau et haut niveau

   iv. Test et débogage

   v. Algorithme et pseudo-code

   vi. Techniques de conception de haut vers le bas et du bas vers le haut

   viii. Paradigme de programmation structurée et paradigme de programmation orientée objet

3. Les langages de programmation ont parcouru un long chemin en termes de niveaux, de générations et de paradigmes. Quels facteurs contribuent à la conception de nouvelles langues

4. Discuter des étapes impliquées dans la résolution de problèmes

5. Divers problèmes effectuent des tâches / fonctions similaires. Expliquer l'importance de ces similitudes dans la programmation.

6. Un algorithme est indépendant du langage. Expliquer.

7. Donner les avantages des organigrammes par rapport aux algorithmes.

8. Énumérer les propriétés d'un bon algorithme.

9. La programmation structurée est un paradigme qui a révolutionné l'industrie de la programmation. Quels sont les avantages de la programmation structurée ?

10. Nommer deux erreurs quelconques qui se dégagent lors de l'exécution du programme.

11. Analyser les deux méthodes principales de test d'un programme.
12. Ecrire un algorithme qui résout le problème suivant :
    i. Évalue le factoriel d'un nombre
    ii. Calcule la moyenne de certains nombres entiers
    iii. Trouve le plus petit nombre d'un tableau d'entiers
    iv. Convertir 12010 en base 2 (système de nombre binaire)
13. Pour les problèmes i à iv en 12 ci-dessus, dessiner un organigramme.
14. Les langages de programmation supportent différents types de données. En utilisant des exemples, discuter quatre types de données quelconques primaires utilisées en langage C.
15. Quelle est l'importance d'apprendre et comprendre les types de données.
16. En utilisant un programme, discuter des étapes suivies lors de l'exécution d'un programme C.
17. Qu'entendez-vous par le terme :
    i. Code objet
    ii. Mot-clé de programme
    iii. Une variable
    iv. Un identificateur
    v. Opérateur d'associativité
18. Nommer les deux parties d'une instruction en langage assembleur
19. Les langages de programmation supportent différents types d'opérateurs. Donner des exemples et discuter cinq types quelconques d'opérateurs.
20. Rédiger des notes courtes sur :
    i. langages de première génération
    ii. langages de deuxième génération
    iii. langages de troisième génération
    iv. langages de quatrième génération

Exercice 2 : Pratique

Instructions

Répondez à toutes les questions en 3 heures, puis consulter le document de référence ou vos notes sur cette unité.

Système de notation

Chaque sous question (i à Vii) de la question 1 est notée sur 2 points si le programme proposé par l'apprenant marche. Ceci est laissé au jugement de l'instructeur.

La question 2 est notée sur 2 points

Enoncé :

1. Ecrire un programme C qui :

   i. Affiche "J'aime la programmation" à l'écran.

   ii. Lire deux entiers, deux nombres réels, un caractère et affiche les à l'écran.

   iii. Divise 7 par 5, affiche la réponse, y compris la partie décimale à l'écran.

   iv. Entrez votre nom et votre âge. Ensuite calculer votre âge (en utilisant un opérateur) et afficher votre âge à votre prochain anniversaire de naissance à l'écran comme suit :

   nom:

   Age:

   DatepProchain anniversaire:

   v. Lire une valeur entière et passer la à une fonction qui renvoie à son tour son factoriel.

   vi. Lire 10 valeurs dans un tableau, évalue la somme des valeurs et affiche le résultat à l'écran.

   vii. Lire votre nom, votre pays, la rue et la ville au clavier et affiche les informations entrés à 'écran comme suit:

**** Ceci est mon adresse ***

* Nom:

* Pays:

* Rue:

* Ville:

2. En utilisation MS-Word, dessiner un organigramme qui calcule la somme des premiers nombres entiers, n étant lu au clavier.

Exercice 3 : Recherche

Instructions

Répondez à toutes les questions en 3 heures, puis consulter le document de référence ou vos notes sur cette unité. Vérifiez votre compréhension en marquant ce que vous avez écrit.

Système de notation

Les points pour cet exercice ne sont pas fournis. Ceci est pour vérifier votre compréhension.

Enoncé :

1. On argumente que normalement, pour atteindre les besoins de réutilisation de code, une conception du programme doit maximiser plus de cohésion avec faible couplage. Discutez les deux concepts : cohésion et couplage.
2. La technique de conception descendante est une technique de résolution de problèmes qui est bien connue dans la programmation. En utilisant une illustration / exemple, discuter cette technique. Comparez-le avec la technique ascendante et pourquoi la stratégie bottom-up n'est pas aussi appréciée que la stratégie top-down.
3. La programmation structurée a trois structures principales. En utilisant des exemples, discuter de ces structures.
4. Chercher pourquoi la programmation orientée objet est devenue populaire dans l'industrie de la programmation. Citer quelques concepts de base de ce paradigme. Comparez-le avec la programmation structurée.
5. Il est conseillé de tester un algorithme avant de le convertir à un code exécutable. Chercher et discuter deux façons de tester un algorithme.
6. Chercher comment un programme peut être écrit à un pourcentage de sortir, par exemple 82%.
7. Chercher comment un programme qui affiche quelque chose en exposant à nombre peut être écrit, par exemple 84 ou St.

Lectures et autres ressources

1. "Achille Braquelaire", Méthodologie de la programmation en C, Norme C 99 - API POSIX, Dunod - 672 pages, 4eédition, 1er mars 2005
2. Yves Mettier",, C en action Solutions et exemples pour les programmeurs en C, Édition : ENI - 653 pages, 2eédition, 7 décembre 2009
3. Fundamentals of programming languages. Dipali P. Baviskar, Technical Publication Pune.
4. Programming and problem solving through "C" languages. Harsha Priya, r. Ranjeet. Firewall media, 2006
5. Programming and PROB solving using C. ISRD, Tata McGraw-Hill education.
6. Programming: Grade in Industrial Technology Engineering. Creative Commons. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf.
7. A brief history of computing. Gerard O'Regan, 2012. Springer London; 2nd edition.
8. Concepts in programming Languages. John C. Mitchell. Cambridge University Press, 2003. Chapter 1.
9. Fundamentals of Programming: with Object Oriented Programming. Gary Marrer. 2009. Gary Marrel.
10. Programming Logic and design, Comprehensive. Joyce Farrell, 2012. 7th edition. Cengage Learning.
11. Problem solving and program design in C. Jeri R. Hanly & Elliot B. Koffman, 6, illustrated, Addison-wesly, 2009.
12. http://archive.oreilly.com/pub/a/oreilly//news/languageposter_0504.html
13. http://www.softpanorama.org/History/lang_history.shtml#General

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