PRINCIPES DE LA PROGRAMMATION INFORMATIQUE

Master **IMSD**
IFRISSE

Pierre Claver OUEDRAOGO
pierreclaver.ouedraogo@gmail.com
+33 7 51 16 24 27

Tables de matières

Aperçu du cours

Vous vous êtes certainement posé la question une fois de savoir comment marche l'ordinateur ? Comment les logiciels qui y sont installés arrivent à comprendre notre besoin?

Prenons par exemple les jeux vidéos, comment arrive-t-on à faire tel ou tel mouvement? Il y avoir une certaine logique derrière tout cela.

Oui ! Et c'est cela l'œuvre de la ***programmation informatique***.

Par conséquent, ce module est 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.

Il faut noter que ce cours est un cours de programmation de débutant conçu pour enseigner les bases de la programmation.

Aperçu du cours

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, etc.

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. Car 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.

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.

Evaluations

Unité 0 : Généralités

Il s'agit d'une unité de pré-évaluation.

Elle va vous familiariser avec certains concepts dont vous avez besoin et qui sont supposés être utilisés dans le module.

Cette unité va vous permettre d'acquérir des compétences sur des notions basiques en informatique , y compris les composants matériels, logiciels et la représentation de données.

Il s'agit ici de vous donner des définitions de certains termes couramment utilisé dans la programmation informatique.

Unité 0 : Généralités
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.

* **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 0 : Généralités
Termes clés

* **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.

Unité 0 : Généralités
Termes clés

* **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).

Unité 0 : Généralités
Termes clés

* **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.

Unité 1. Introduction à la programmation

Les ordinateurs sont partout. A travers les programmes qui y sont installés , Ils aident l'humain dans l'administration efficace des tâches dans tout les domaines: Administration publiques, privées, dans le domaine de la santé , de la défense , dans le commerce ,etc.

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 .

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:

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

Unité 1: 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.

* **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.

Unité 1: Termes clés

* **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).

Unité 1: Histoire de la programmation

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.

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

Unité 1: Histoire de la programmation

* 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.
* Année 1980: C++, Objective-c, Perl,etc..
* Année 1990 - 2000: Visual Basic, Java, PHP, JavaScript;
* De nos jours : C#, Scala, Swift, Ceylon

Référence : https://fr.wikipedia.org/wiki/Chronologie_des_langages_de_programmation

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

Les langages de programmation ont évolué au fil du temps. Dans les années 1940 à 1950 les premiers langages également langages de première génération utilisaient le code machine pour programmer l'ordinateur. 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.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

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.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

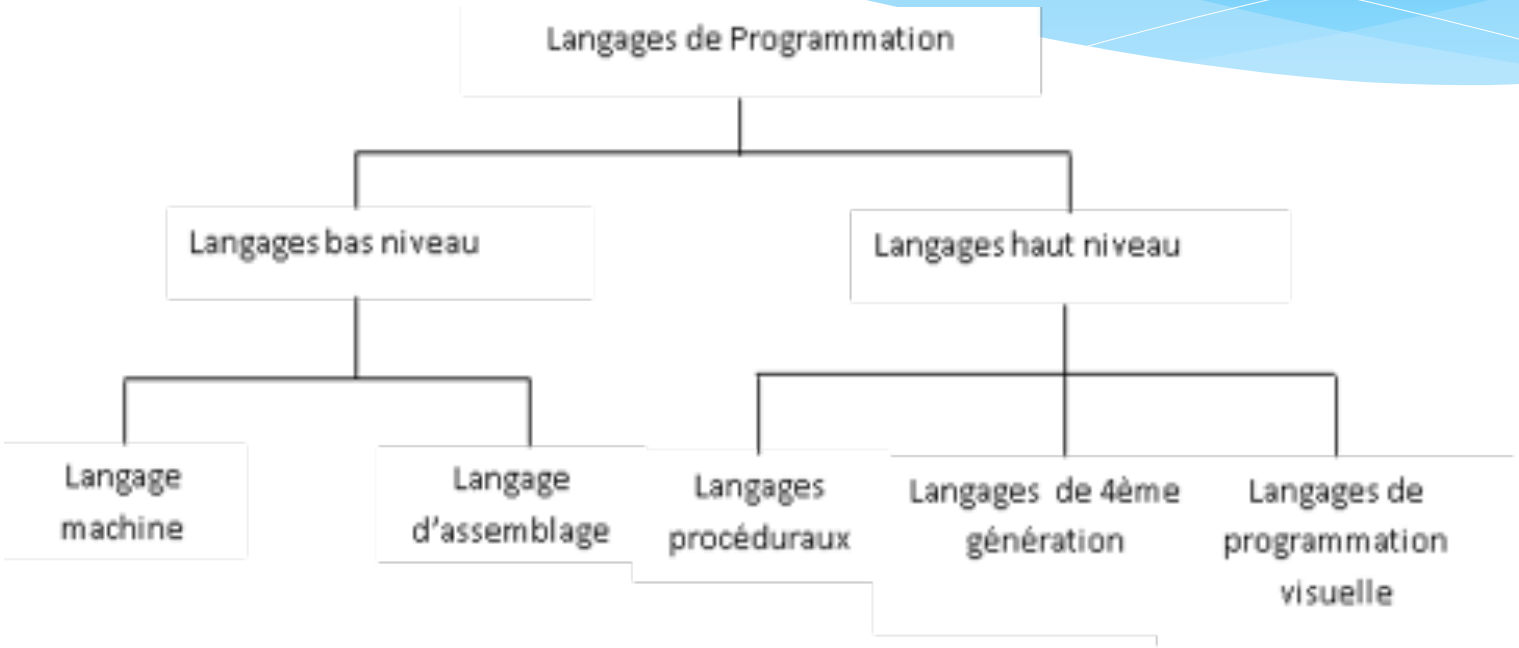


Figure 1.1 : Les niveaux de langages de programmation

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.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

Langage machine: En utilisant ce langage, les programmes sont écrits en utilisant des 0 et des 1. On parle ici du binaire. 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 débuguer.

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.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

Langages de haut niveau: Les programmes sont écrits en anglais comme des déclarations. Ils ne sont pas directement exécutables donc des **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(4è génération) et soulignent sur ce qui doit être accompli plutôt que la façon de l'accomplir. Comme exemple nous avons Oracle, SQL, etc.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

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.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

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. Ce qui fait que les compilateurs sont plus rapides que les interpréteurs.

Unité 1: 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.

Unité 1: Les niveaux, les générations et les paradigmes des langages de programmation

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.

Unité 2: Résolution de problèmes basés sur l'informatique

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.

Ainsi la programmation implique l'automatisation des systèmes, des processus, des opérations ou une activité pour aboutir à une solution (programme) qui 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 et la conception de la solution.

Unité 2: Objectifs de l'unité

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

- Formuler et définir les problèmes.
- Représenter des solutions en utilisant des algorithmes et pseudocodes.
- Décrire les caractéristiques d'un algorithme et d'un pseudocode.
- Décrire les techniques de programmation.

Unité 2: 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 y a des entrées). En d'autres termes, un algorithme est une procédure d'étape par étape pour résoudre un problème donné.

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.

Unité 2: Termes clés

Modèle: est une abstraction de la réalité.

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

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.

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

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.

Unité 2: Termes clés

Pseudocode: En programmation, le **pseudo-code**, également appelé **LDA** (pour **Langage de Description d'Algorithmes**) est une façon de décrire un algorithme en langage *presque naturel*, sans référence à un langage de programmation en particulier.

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: Il s'agit de rechercher les spécifications du problème par le biais de questionnaires, des entretiens et d'observations entre autres méthodes.

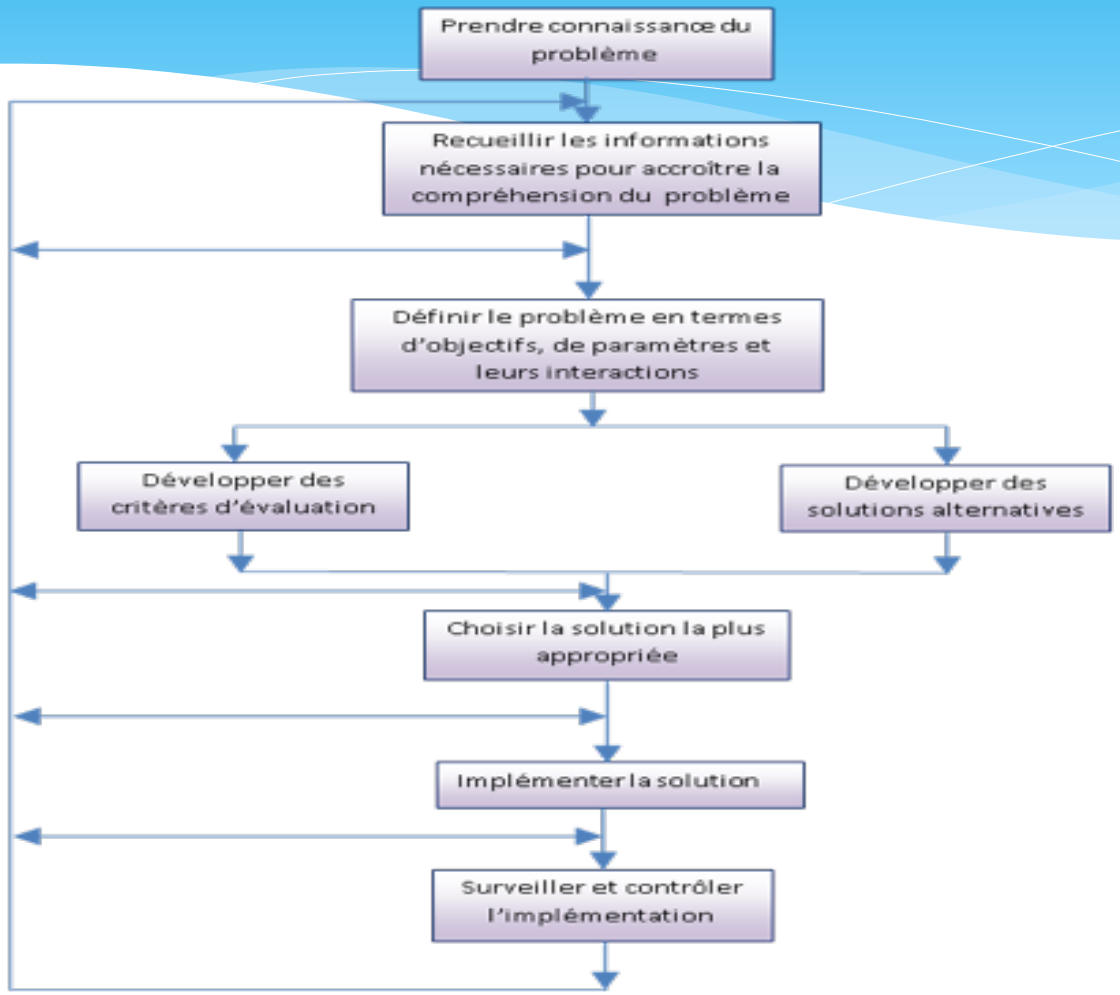
Cycle de Vie de Développement d'un Logiciel: C'est la méthodologie ou processus utilisé par les personnes qui travaillent dans la technologie de l'information pour résoudre des problèmes en fournissant une solution efficace et efficiente.

Unité 2: Résolution de problèmes

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

Unité 2: Résolution de problèmes



Unité 2: Conception de programme

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 qui est l'un des éléments de base de la programmation structurée.

Nous allons aborder dans cette partie du cours :

- Les algorithmes et
- Les organigrammes

Unité 2: 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

Unité 2: Algorithmes

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.

Unité 2: Algorithmes

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

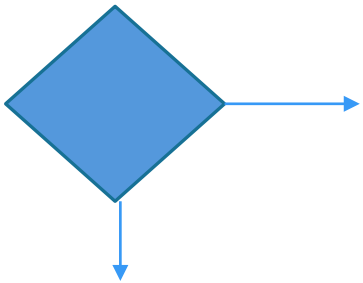
Unité 2: Algorithmes

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

Début / Fin :

Traitement :

Décision :



Boucle :

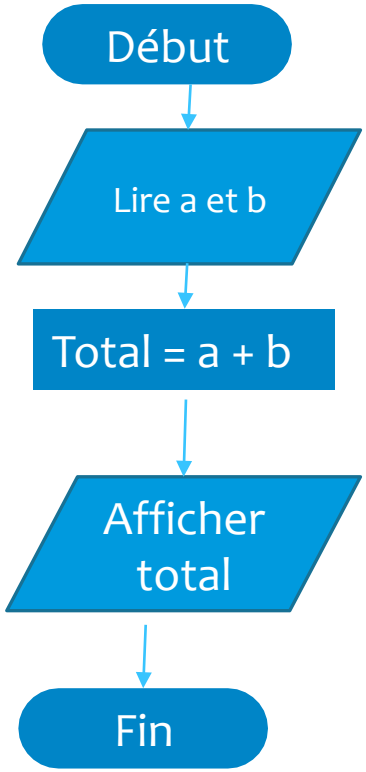
Entrée/Sortie :

Connecteur :

Unité 2: Algorithmes

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

Exemple d'organigramme



Unité 2: Types de 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.

La conception modulaire des programmes : En informatique, la programmation modulaire reprend l'idée de fabriquer un produit (le programme) à partir de composants (les modules).Elle décompose une grosse application en modules, groupes de fonctions, de méthodes et de traitement, pour pouvoir les développer et les améliorer indépendamment, puis les réutiliser dans d'autres applications

Exercices

Exercice1 : 5 min

Écrire un algorithme qui demande en entrée un nombre à l'utilisateur puis calculer le carre et afficher le carré de ce nombre.

Exercice2 : 5min

Écrire un algorithme permettant d'afficher la saison (pluvieuse , sèche) en introduisant le numéro du mois.

Corrigé

* Exercice 1 :Algorithme Carre

Début

Ecrire « veuillez saisir un nombre »

Lire(X)

X*X

Afficher X*X

Fin.

* Exercice 2 : AlgorithmeSaison;

```
VarM:entier;
Début
    Ecrire('Donner un numéro de mois 1--12');
    RépéterLire(M);Jusqu'à M>0 et M<13

        Cas M Vaut
            7,8,9,10: Ecrire('La saison est: saison pluvieuse');
            1,2,3,4,5,6,11,12:  Ecrire('La saison est: saison sèche');
        FinCas;
Fin.
```

Unité 3: Programmation en langage C

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.

Dans cette unité , nous allons aborder les concepts de programmation comme les variables, les types de données, les opérateurs, les structures de contrôle, les tableaux et les procédures ou fonctions

Unité 3: Objectifs de l'unité

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

- Distinguer les différents types de données.
- Fournir des solutions en écrivant des programmes simples en C.

Unité 3: 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).

Unité 3: Termes clés

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.

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

Unité 3: Termes clés

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**

Unité 3: Termes clés

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 .

Une bibliothèque en C est un groupe de fonctions et de déclarations, exposées pour une utilisation par d'autres programmes.

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 (//) .

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

Unité 3: Termes clés

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 **.**

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.

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

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

Unité 3: Termes clés

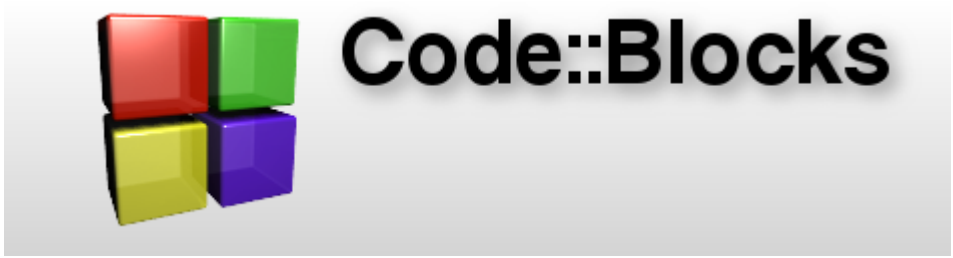
Les variables locales: Les variables qui sont déclarées dans une fonction ou d'un bloc sont appelées variables locales. Ils peuvent être utilisés que par des déclarations qui sont à l'intérieur de cette fonction ou de bloc de code. (http://www.tutorialspoint.com/cprogramming/c_ scope_rules.htm)

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)

Unité 3: Programme C

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.

Beaucoup de compilateurs C existent sur le marché et ont été conçus ou développés par différents groupes de développeurs de programmes. Dans ce cours , nous allons utiliser **CodeBlocks** compte tenu de sa facilité de prise en main.



Vous pouvez le télécharger et l'installer en allant sur le site http://www.codeblocks.org/downloads/25 puis **Download the source code** ou directement sur :

http://sourceforge.net/projects/codeblocks/files/Sources/20.03/codeblocks-20.03.tar.xz

Unité 3: Processus d'exécution de source de programme C

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

- Création du programme
- Compilation du programme
- Lier le programme avec les fonctions qui sont nécessaires à partir de la bibliothèque C
- L'exécution du programme

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.

Unité 3: Programme C

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.

Unité 3: Programme C

Structure d'un programme C

```
#include <stdio.h>
#include <stdlib.h>

/* run this program using the console pauser or add your own
getch, system("pause") or input loop */

int main(int argc, char *argv[]) {
printf("hello");
return 0;

}
```

Fichiers d'entête

un commentaire

Fonction principale

Corps du programme contenant les instructions exécutables contenues entre '{ }'.

Unité 3: Programme 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.

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.

Ci-dessous un exemple de 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 :

Unité 3: Exemple de programme C

```
#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 leclavier
printf("y=%d\n",y); // impression de la valeur de y
printf("age=%d",age); // impression de âge
return 0;
}
```

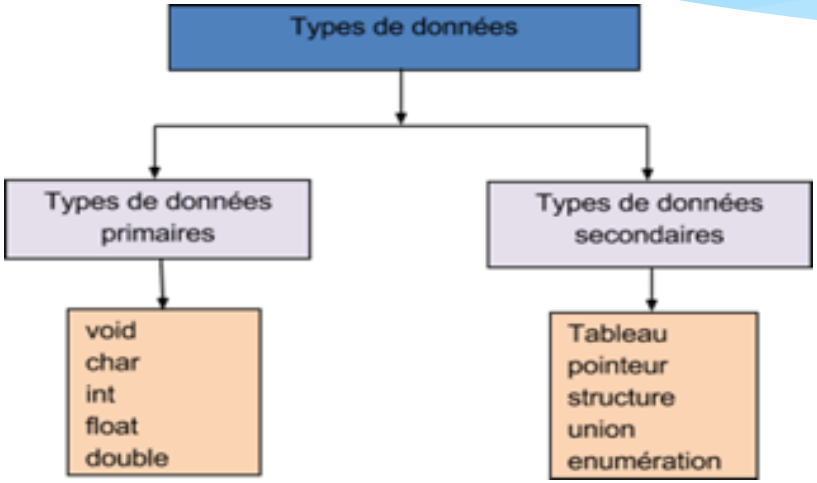
Unité 3: Les types de données

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.

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

Unité 3: Les types de données

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.

Unité 3: Les 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 :

Unité 3: Les variables

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),

Unité 3: Les variables

C supporte un certain nombre de constantes qui sont:

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)

Les constantes réelles

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

constantes de caractères

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

Unité 3: Les variables

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.

Unité 3: Les 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.

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 :

Unité 3: Les Opérateurs

Les opérateurs arithmétiques

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

Les opérateurs relationnels

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

Unité 3: Les Opérateurs

Les opérateurs logiques

OPERATEUR	SIGNIFICATION
&&	ET
||	OU
!	Non

L'opérateur d'affectation :il est représenté en utilisant un opérateur '=' par exemple x = 6;

Les opérateurs d'incrémentation et de décrémentation

OPERATEUR	SIGNIFICATION
++	Incrémentation
--	Décrémentation

L'opérateur conditionnel : C'est un opérateur ternaire qui contient trois opérandes et sa syntaxe ressemble à ceci : **exp1 ? exp2 : exp3;**

Unité 3: Les Opérateurs

L'opérateur virgule

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

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

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

Unité 3: Les Opérateurs

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.

Unité 3: Les Opérateurs

Expressions

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.

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

Unité 3: Structures de contrôle et Fonctions

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.**

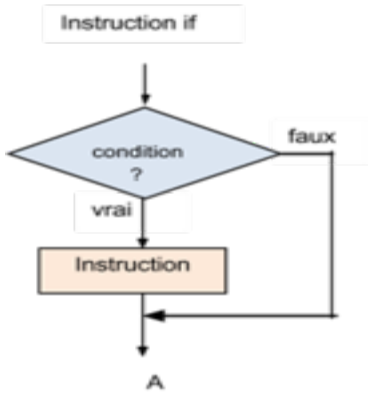
Les branchements sont également connus comme structures conditionnelles ou décisionnels.

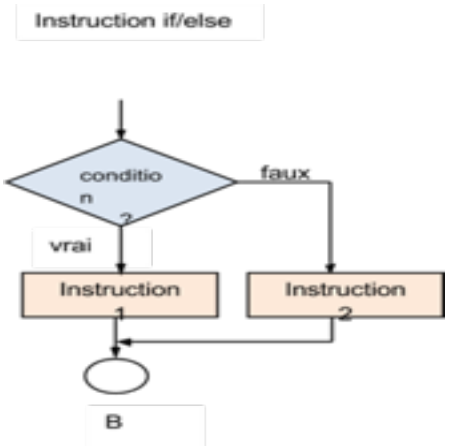
Les structures de boucle répètent une commande tant que la condition est remplie / vrai. Ces deux structures de contrôle font partie des composants de la programmation structurée (Cf. unité 2) qui rendent un programme plus lisible et compréhensible.

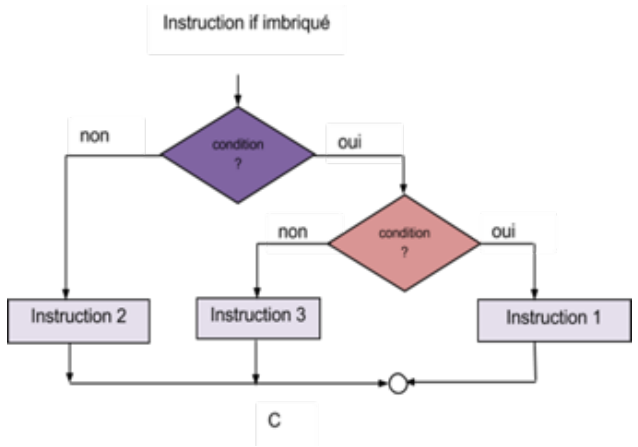
Unité 3: Structures de contrôle et Fonctions

Structures de contrôle conditionnelles

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). On peut également avoir une instruction if imbriqué (voir figure C).







Unité 3: Structures de contrôle et Fonctions

Instruction Switch

C'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;
}
```

Unité 3: Structures de contrôle et Fonctions

L'opérateur: (Opérateur conditionnel)

Cet opérateur fonctionne comme l'instruction if/else.

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;
}
```

Unité 3: Structures de contrôle et Fonctions

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. Ce sont des instructions de boucle. En exemple on a :

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.

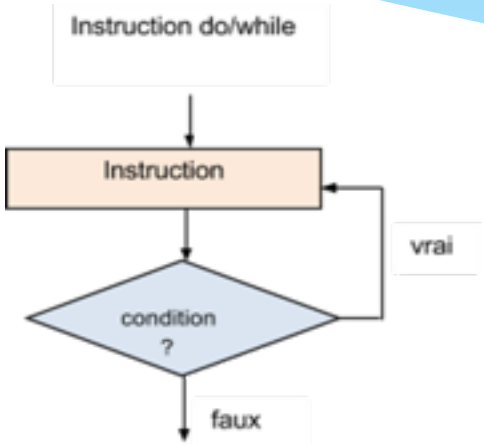
```
while (condition)
{
instruction(s);
}
```

for: Tout comme l'instruction while, il répète une boucle un certain nombre de fois.

```
for(initialisateur; condition; incrémentation)
{
instruction(s);
}
```

Unité 3: Structures de contrôle et Fonctions

L'instruction do/while



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 que 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.

Unité 3: Structures de contrôle et Fonctions

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

Une fonction est un groupe d'instructions qui effectuent une tâche particulière. En termes simples, toutes les instructions dans un groupe spécifique qui 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.

Unité 3: Structures de contrôle et Fonctions

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.

Unité 3: Structures de contrôle et Fonctions

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. Par exemple :

```
#include <stdio.h>
void salutation (void)
{
printf (“Bonjour!”);
}
main ()
{
salutations (); // Appel de la fonction
}
```

Unité 3: Structures de contrôle et Fonctions

Par exemple :

```
#include <stdio.h>
void salutation (void)
{
printf ("Bonjour!");
}
main ()
{
salutations (); // Appel de la fonction
}
```

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.

Unité 3: Structures de contrôle et Fonctions

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

Unité 3: Les tableaux et les chaînes de caractères

Les tableaux

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 :

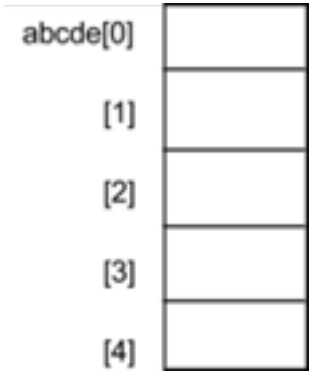
int abcde[5];

Float balance[10];

Unité 3: Les tableaux et les chaînes de caractères

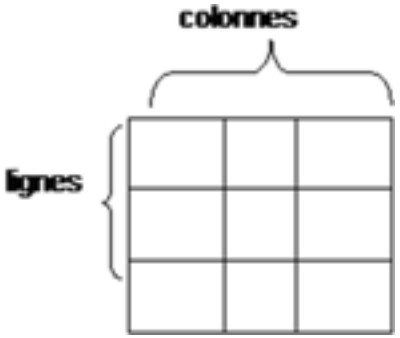
Les éléments du tableau sont accessibles en utilisant le nom du tableau et l'indice de l'élément. Les éléments de tableau sont stockés de manière contiguë et le premier élément du tableau commence avec un indice zéro (0), le dernier élément du tableau se termine avec un indice égal à la taille du tableau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

Un tableau peut être initialisé comme:



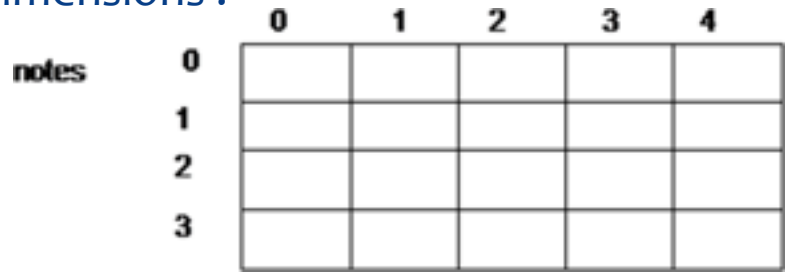
Unité 3: Les tableaux et les chaînes de caractères

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 [nbLigne] [nbCol];



Unité 3: Les tableaux et les chaînes de caractères

Chaînes 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];

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','\o'};

ou : char prenom[10]="Antilolas";

Unité 4: Notions de test de programme et de Débogage

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 permettra à l'apprenant de se familiariser aux concepts essentiels (tests, débogage) qui sont communs à tous les langages et paradigmes de programmation.

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.

Unité 4. 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

Unité 4. Termes clés

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

Unité 4. Erreurs de programme et test

Il existe trois types d'erreurs de base à savoir :

- syntaxe ou erreurs de compilation;
- erreurs d'exception à l'exécution;
- 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.

Unité 4. Erreurs de programme et test

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

Unité 4. Débogage de programme

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.

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.

Unité 4. Débogage de programme

L'IDE comprend un débogueur, qui est un outil de support pour trouver des erreurs d'exécution.

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.

Unité 4. Différence entre Test et Débogage de programme

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

Unité 5. Documentation de programme/logiciel

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.

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.

Exercices

Conclusion

Ce cours a fait ressortir les différentes notions et bases nécessaires à la programmation informatique .Vous êtes maintenant à 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.

Cependant réussir en programmation nécessite beaucoup de pratique. Il faut donc vous performer en faisant beaucoup de travaux personnels .