REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE SUPERIEURE DES SCIENCES APPLIQUEES
–T L E M C E N–

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

DÉPARTEMENT DE LA FORMATION PRÉPARATOIRE

POLYCOPIÉ Des TRAVAUX DIRIGES ET TRAVAUX PRATIQUES

Informatique I

Élaboré Par :

Dr. BEKKAOUI Mokhtaria

Sommaire

Rappel cours

Travaux Dirigés

Travaux Pratiques

Table des figures & des tableaux

Préface

Ce polycopié, regroupe un ensemble des travaux dirigés et pratiques destinés aux élèves ingénieurs de la première année tronc commun, afin de leurs donner les principes de base concernant l'architecture de l'ordinateur et de la programmation en langage C. Le contenu de ce polycopié est adapté avec le programme du canevas du socle commun.

Ce polycopié est organisé en deux parties, la première est réservée pour le codage, algèbre de BOOLE et introduction à l'algorithmique. Alors que la deuxième partie est planifiée pour concevoir un programme utilisant des techniques structurées de développement avancée en langage C. Chaque partie inclut un rappel sur les concepts étudiés.

Afin d'assurer le bon déroulement de ces travaux (dirigés et pratiques), les étudiants doivent assurer leur cours pour en ensemble des prérequis qui s'appuient sur des connaissances de base en informatique.

En fin, j'espère que ce polycopié pourra apporter un plus et une assistance pédagogique aux enseignants chargés de TD ou TP, ainsi qu'aux étudiants dans leur formation en informatique.

Rappel cours

Informatique I : Rappel cours

Objectifs

• Connaitre les propriétés des principaux codages des entiers.

• Connaitre les aspects théoriques et pratiques de l'analyse, de la synthèse et de la matérialisation de circuits logiques qu'on trouve dans les ordinateurs.

•Avoir un aperçu sur l'architecture d'un ordinateur.

• Définir un algorithme permettant de résoudre le problème.

• Savoir transcrire cet algorithme dans un langage cible.

- Planifier et concevoir un programme utilisant des techniques structurées de développement.
- Prévoir, concevoir, créer et employer les fonctions en décomposant un problème en sous-tâches.
- Passer des arguments par référence ou par valeur entre fonctions. Différentes dimensions.
- Écrire des instructions de programmation valides pour déclarer, initialiser, manipuler et passer les pointeurs comme arguments aux fonctions.
- Utiliser et expliquer la relation entre les pointeurs et les valeurs qu'ils indiquent
- Utiliser et manipuler les structures de données
- Utiliser les outils du langage C pour l'implantation des solutions algorithmiques

Chapitre 1 : Représentation des nombres

1.1. Systèmes de numération

Comprendre c'est quoi un système de numération

- Apprendre la méthode de conversion d'un système à un autre
- Apprendre à faire des opérations arithmétiques en binaire.

Dans un système de numérotation en base B, un nombre noté (B) égal à :

$$N_{(B)} = \sum_{k=0}^{n-1} a_k B^k = a_{n-1}a_{n-2} \dots a_2 a_1 a_0$$

Avec :
B : base ou nombre de chiffres différents qu'utilise le système de numérotation.
a_k : Chiffre de rang k

B_k : Pondération associée à a_k

De nombreux systèmes de numération sont utilisés en électronique numérique. Les plus courants sont les systèmes de numération suivants :

- Binaire (Base 2) : La numération binaire (ou base 2) utilise deux symboles appelés BIT (Binary digIT) : 0 et 1

- Décimal (Base 10) : dispose de dix symboles (en l'occurrence des chiffres) qui sont:{0, 1, 2, 3, 4, 5, 6, 7, 8,9}

- Octal (Base 8) : Ce système tend aujourd'hui à disparaître au profit de la base 16 suite à l'évolution technologique des composants. Anciennement, il servait au codage des nombres dans les ordinateurs de première génération. Il utilise 8 symboles : 0, 1, 2, 3, 4, 5, 6, 7.

- Hexadécimal (Base 16) : Ce système de numération est très utilisé dans les systèmes ordinateurs et micro-ordinateurs ainsi que dans le domaine des transmissions de données. Il comporte 16 symboles les chiffres de 0 à 9 et les lettres {A, B, C, D, E, F}.

1.2. Conversions

1.2.1. Conversion d'un nombre décimal vers une base quelconque [$(N)_{10}$ -> $(N)_B$]

Pour convertir un nombre de la base 10 vers une base B quelconques, il faut faire des divisions successives par B et retenir à chaque fois le reste jusqu'à l'obtention à un quotient inférieur à la base B, dans ce cas le nombre s'écrit de la gauche vers la droite en commençant par le dernier quotient allant jusqu'au premier reste.

1.2.2. Conversion d'un nombre d'une base quelconque vers la base décimal [$(N)_B$ -> $(N)_{10}$]

Pour ce type de conversion, il suffit de représenter le nombre par une combinaison linéaire des puissances successives de la Base et faire la somme, le résultat ainsi trouvé s'écrit directement dans la BASE 10.

Base de départ	Base d'arrivée	Méthode de transcodage
Décimal	Binaire	Méthode de la division par 2 du nombre
	Octal	Méthode de la division par 8 du nombre
	Hexadécimal	Méthode de la division par 16 du nombre
Binaire	Décimal	$(N)_{10} = a_n B^n + a_n B^n + \cdots + a_0 B^0 \; avec \; B = 2$
Octal		$(N)_{10} = a_n B^n + a_n B^n + \cdots + a_0 B^0 \; avec \; B = 8$
Hexadécimal		$(N)_{10} = a_n B^n + a_n B^n + \cdots + a_0 B^0 \; avec \; B = 16$

2.3. Conversion d'un nombre en base B vers une base quelconque B' [$(N)_B$ -> $(N)_{B'}$]

• binaire <=> hexadécimal : $2^4 = 16$ => un groupe binaire de 4 bits est transcodable directement en un digit hexadécimal. 0n divise le nombre binaire en tranches de 4 bits (à partir du LSB). Chacun des quartets est ensuite converti en un digit hexadécimal par simple sommation pondérée.
• binaire <=> Octal : $2^3 = 8$ => c'est la même méthode pour passer de binaire ver hexadécimal sauf qu'on divise le nombre binaire en tranches de 3 bits (à partir du LSB).

1.3. Représentation des nombres signes en binaire

Il existe 3 conventions pour exprimer les nombres signés dans le système binaire :
- *Représentation signe et valeur absolue (binaire signé)* : L'une des méthodes est de réserver un bit pour indiquer le signe du nombre, d'où l'appellation de binaire signé. Le bit réservé au signe est toujours le bit le plus à gauche. Pour le bit de signe et par convention, le 0 représente le (+) et le 1 le (-).
- *Représentation en complément à 1 :* Pour calculer le complément à 1(CA1) d'un nombre binaire, il suffit de complémenter chaque bit de ce nombre c'est-à-dire remplacé les 1 par des 0 et les 0 par des 1.
- *Représentation en complément à 2 :* Pour calculer le CA2 d'un nombre binaire N, on ajoute la valeur 1 au CA1 de N. (CA2 = CA1 + 1)

1.4. Arithmétiques binaire

Addition : L'addition de 2 nombres binaires est parfaitement analogue à l'addition de 2 nombres décimaux. Il faut commencer par le bit de poids le plus faible en utilisant l'algorithme suivant :

0 + 0 = 0 1 + 0 = 1
1 + 1 = 10 (= 0 + report de 1 sur la gauche) 1 + 1 + 1 = 11 (= 1 + report de 1 sur la gauche)

Soustraction : Dans le cas de la soustraction de deux nombres binaires non signés on peut utiliser l'algorithme suivant :
0 - 0 = 0 0 - 1 = 1 (avec un report de 1 à retrancher au chiffre supérieur)
1 - 0 = 1 1- 1 = 0

Chapitre 2: Algèbre de BOOLE

En général, l'algèbre de Boole est une structure mathématique qui permet d'exprimer le fonctionnement de tout système logique à deux états. Les conditions y sont représentées par des variables et les relations par des signes et on peut relier variables et relations sous forme d'équations. De plus il existe des règles qui permettent de réduire les équations.

2.1. Expression Booléenne

Une variable logique est une grandeur qui peut prendre deux valeurs qui sont repérées habituellement 0 ou 1. Se note par une lettre comme en algèbre.

Une expression logique ou bien une fonction logique est le résultat de la combinaison (logique combinatoire) d'une ou plusieurs variables logiques reliées entre elles par des opérations logique de base (ET, OU, NON).

Une fonction logique possède une ou des variables logiques d'entrée et une variable logique de sortie exemple : $f(x, y) = x.y + \bar{x}.y + \bar{y}$

2.2. Table de Vérité

Les fonctions logiques peuvent être représentées par des Tables de vérités, elle permet la connaissance de la sortie (d'un circuit logique) en fonction des diverses combinaisons des valeurs des entrées

- Le nombre de colonnes est le nombre total d'entrées et de sorties
- Le nombre de lignes est 2^N sachant que "N" est le nombre d'entrées,

Exemple:

Une fonction de 3 entrées et 1 sortie se représente par une table de 4 colonnes et 8 lignes.

Exemple :

A	B	C	f (A, B, C)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

2.3. Porte Logique

Avec l'évolution de la technologie, on inventa des composants plus versatiles que les circuits à relais, qui donnèrent naissance aux circuits logiques. On définit alors un ensemble de composants appelés portes logiques.

Chaque porte correspond à une fonction logique précise, à laquelle on associe un symbole. Les portes élémentaires sont :

Opérateur logique	Nom français	Nom anglais	Symbole	Table de vérité
$A \cdot B$	ET	AND		A B F 0 0 0 0 1 0 1 0 0 1 1 1
$A+B$	OU	OR		A B F 0 0 0 0 1 1 1 0 1 1 1 1
$\overline{A \cdot B}$	NON ET	NAND		A B F 0 0 1 0 1 1 1 0 1 1 1 0
$\overline{A+B}$	NON OU	NOR		A B F 0 0 1 0 1 0 1 0 0 1 1 0
$A \oplus B$	OU exclusif	XOR		A B F 0 0 0 0 1 1 1 0 1 1 1 0
$\overline{A \oplus B} = A \otimes B$	NON OU exclusif	XNOR		A B F 0 0 1 0 1 0 1 0 0 1 1 1
$\overline{A}$	NON (inverseur)	NOT (inverter)		A F 0 1 1 0

2.4. Circuit Logique

Le passage d'une fonction logique à un circuit logique induit certaines modifications aux notions de l'algèbre de Boole. Les variables d'une fonction logique deviennent les entrées du circuit, ce circuit donnant en sortie la valeur de la fonction logique suivant la valeur des entrées. Ainsi, il devient possible de connecter des portes logiques les unes aux autres pour réaliser une fonction logique; et de manière symétrique, trouver la fonction logique réalisée par un circuit nous permet de le manipuler en vue de simplifications éventuelles.

En pratique : circuit électrique (transistors) dans lequel une faible tension représente le signal 0 alors qu'une tension élevée correspond au signal 1.

Exemple : $\boldsymbol{L(X,Y,Z) = \bar{X}Z + XY}$

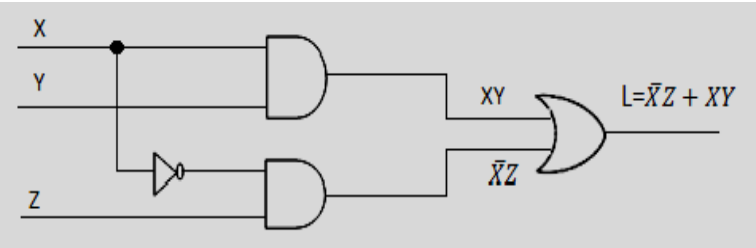


Figure 1 Circuit électronique

2.5. Simplification des expressions booléennes

Simplifier = diminuer le nombre d'opérateurs = diminuer le nombre de portes logiques (et donc le coût).

Deux méthodes pour la simplification :

- Algébrique (en utilisant des propriétés et des théorèmes voir le cours)
- Graphique (tableaux de Karnaught; ...)

La table de Karnaugh est un outil graphique qui permet de simplifier de manière méthodique des expressions booléennes.

Les règles pour la simplification des fonctions booléennes avec le diagramme de Karnaugh son t les suivantes :

Entourer tous les 1 dans des rectangles :

- les plus grands possibles,
- tels que leur taille est une puissance de 2,
- éventuellement sur les bords.

En déduire la formule et le circuit :

- une somme (OU) des formules de chaque rectangle
- le formule d'un rectangle est un produit (ET) :

  Des variables qui valent toujours 1 dans ce rectangle

  Des négations de celles qui valent toujours 0.

  Les autres variables n'apparaissent pas dans le ET.

Chapitre 3: La machine de Von Neuman

Dans cette partie, nous décrivons rapidement l'architecture de base d'un ordinateur et les principes de son fonctionnement.

Un ordinateur est une machine de traitement de l'information. Il est capable d'acquérir de l'information, de la stocker, de la transformer en effectuant des traitements quelconques, puis de la restituer sous une autre forme. Le mot informatique vient de la contraction des mots information et automatique.

Les termes Hardware et Software sont très courants en informatique.

3.1. Architecture matérielle (Hardware)

Le terme Hardware signifie « matériel » en français. Il englobe notamment tous les composants physiques d'un ordinateur. En d'autres termes, c'est tout ce qui constitue celui-ci. Les composants Hardware sont des matériels physiques palpables, pouvant être à l'intérieur ou à l'extérieur de votre PC. Les deux principaux constituants d'un ordinateur sont la mémoire principale et le processeur. La mémoire principale (MP en abrégé) permet de stocker de l'information (programmes et données), tandis que le processeur exécute pas à pas les instructions composant les programmes.

Le processeur est capable d'exécuter des programmes en langage machine, c'est à dire composés d'instructions très élémentaires suivant un codage précis. Chaque type de processeur est capable d'exécuter un certain ensemble d'instructions, son jeu d'instructions.

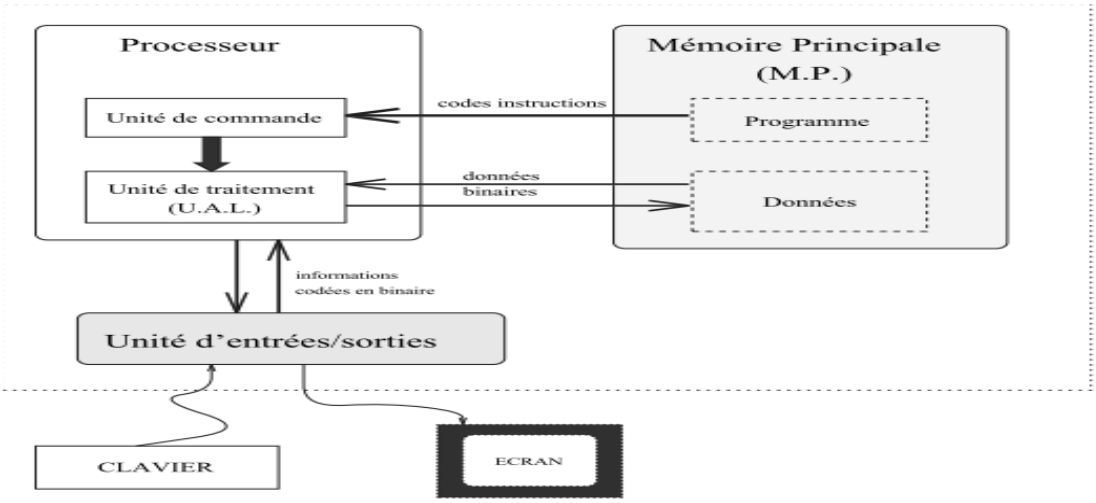


Figure 2 Architecture Von Neuman

- ***Mémoire (Programme et donnée)*** : Dispositif électronique qui stocke les programmes et les données.
- ***Unité de contrôle*** : Dirige les autres unités en utilisant l'horloge et les signaux de contrôle.
- ***Unité arithmétique et logique*** : Exécute les opérations arithmétique et logique telle que : l'addition, la soustraction, la multiplication et la division.
- ***Entrée / Sortie*** : Entrée : Reçoit les entrées (programmes et données) de l'utilisateur
  Sortie : Envoie-les sorties vers l'utilisateur.

Pour chaque instruction, le processeur effectue schématiquement les opérations suivantes :
1. lire en mémoire (MP) l'instruction à exécuter ;
2. effectuer le traitement correspondant ;
3. passer à l'instruction suivante.

Le processeur est divisé en deux parties, l'unité de commande et l'unité de traitement :
– l'unité de commande est responsable de la lecture en mémoire et du décodage des instructions;
– l'unité de traitement, aussi appelée Unité Arithmétique et Logique (U.A.L.), exécute les instructions qui manipulent les données.

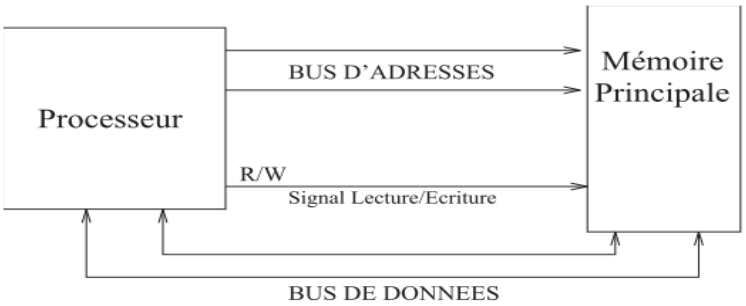


Figure 3 Connexions Processeur-Mémoire : bus de données, bus d'adresse et signal lecture/écriture.

Les informations échangées entre la mémoire et le processeur circulent sur des bus. Un bus est simplement un ensemble de n fils conducteurs, utilisés pour transporter n signaux binaires.
Le bus d'adresse est un bus unidirectionnel : seul le processeur envoie des adresses. Il est composé de a fils ; on utilise donc des adresses de a bits. La mémoire peut posséder au maximum 2^a emplacements (adresses 0 à $2^a - 1$).
Le bus de données est un bus bidirectionnel. Lors d'une lecture, c'est la mémoire qui envoie un mot sur le bus (le contenu de l'emplacement demandé) ; lors d'une écriture, c'est le processeur qui envoie la donnée.

3.2. Les Logiciels (Software)

Un logiciel ou programme ou encore application est un ensemble d'instructions informatiques permettant l'exécution des tâches par l'ordinateur.

3.2.1. Système d'exploitation

Le système d'exploitation (Microsoft Windows, Linux, MacOs, etc) est le logiciel principal de l'ordinateur qui permet de coordonner les éléments de l'unité centrale. Il contrôle l'ordinateur et ses périphériques et permet de lancer les applications.
Le système d'exploitation est multitâche ce qui signifie qu'il peut lancer plusieurs applications simultanément. On retrouve généralement Windows et Linux sur les PC, les Mac disposant de leur propre système d'exploitation (MacOs).

3.2.2. Pilotes

Un pilote (driver en anglais) est un logiciel qui permet au système d'exploitation d'identifier et d'assurer le fonctionnement d'un matériel sur l'ordinateur. Les systèmes d'exploitation intègrent un grand nombre de pilotes permettant de faire fonctionner sans installation supplémentaire un grand nombre de composants, on parle de pilotes génériques.

3.2.3. Logiciel applicatif

C'est un logiciel dont les automatismes sont destinés à assister un utilisateur dans une de ses activités (comptabilité, traitement de texte, retouche photo, jeux vidéo …). Exemples : OpenOffice Writer, Gimp, la calculatrice de Windows, etc.

Chapitre 4: Introduction à l'algorithmique

Malgré que les langages soient de plus en plus proches du langage humain, ils ne sont pas directement lisibles. C'est pourquoi, dans ce qui suit, nous allons utiliser un pseudo-langage (Algorithme), comportant toutes les structures de base d'un langage de programmation. Il suffira ensuite de traduire notre "pseudo" en langage évolué en fonction des possibilités de ce langage.
Un algorithme est une séquence finie d'instructions claires, discrètes et non ambigües pour résoudre un problème particulier.

- Il reçoit des données en entrées, et produit des données en sortie.
- Chaque instruction nécessite un temps bien déterminé pour être exécuter.

En bref, un algorithme est une série d'instructions qui vont résoudre un problème (tache complète).

4.1. Types numériques classiques

Tous les langages, quels qu'ils soient offrent un « bouquet » de types numériques, dont le détail est susceptible de varier légèrement d'un langage à l'autre. Grosso modo, on retrouve cependant les types suivants :

Type Numérique	Plage
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	$-3{,}40\times10^{38}$ à $-1{,}40\times10^{45}$ pour les valeurs négatives $1{,}40\times10^{-45}$ à $3{,}40\times10^{38}$ pour les valeurs positives
Réel double	$1{,}79\times10^{308}$ à $-4{,}94\times10^{-324}$ pour les valeurs négatives $4{,}94\times10^{-324}$ à $1{,}79\times10^{308}$ pour les valeurs positives

Opérateurs numériques :

Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique.

+ : addition

- : soustraction

* : multiplication

/ : division

Mentionnons également le ^ qui signifie « puissance ».

4.2. L'instruction d'affectation

- Dans la partie gauche du =, on met le nom de la variable et dans la partie droite on met la valeur ou bien l'expression.
- Chaque variable fait référence à une unique location dans la mémoire de l'ordinateur qui contient une certaine valeur. Une affectation est exécutée en deux étapes :

• évaluation de l'expression de la partie droite.

• attribuer la valeur résultante (returned value) à la cellule de la mémoire correspondant à la variable de la partie gauche. Exemple X = 5Y + 16

4.3. Les instructions de lecture et d'écriture

Lecture et Ecriture sont des instructions algorithmiques qui ne présentent pas de difficultés particulières, une fois qu'on a bien assimilé ce problème du sens du dialogue (homme → machine, ou machine ← homme).

Lecture : Lire (variable) La variable doit être des données. Ici l'utilisateur donne une valeur (par clavier). Cette valeur donnée est affectée à la variable.

Ecriture : Ecrire (variable) La variable doit être des résultats. La valeur de la variable est affichée (sur écran).

4.4. Les Tests

Il n'y a que deux formes possibles pour un test ; la première est la plus simple, la seconde la plus complexe.

Si booléen **Alors debut_si**
Instructions
Finsi
Si booléen **Alors debut_si**
Instructions 1
Finsi
Sinon debut_sinon
Instructions 2
Finsinon

(Si… Alors) la machine examine la valeur du booléen. Si ce booléen a pour valeur VRAI, elle exécute la série d'instructions. Cette série d'instructions peut être très brève comme très longue, cela n'a aucune importance. En revanche, dans le cas où le booléen est faux, l'ordinateur saute directement aux instructions situées après le Finsi.

4.5. Les Boucles

Un programme a presque toujours pour rôle de répéter la même action un certain nombre de fois. Pour ce faire, nous utiliserons une structure permettant de dire " Exécute telles actions jusqu'à ce que telle condition soit remplie"
Bien qu'une seule soit nécessaire, la plupart des langages de programmation proposent trois types de structure répétitive.

- Tantque

Tantque booléen **début_tantque**
Instructions
FinTantQue

Ce qui signifie : Tant que la condition est vraie, on exécute les instructions.

- Pour

Très souvent, nous utilisons une structure répétitive avec un compteur et nous arrêtons lorsque le compteur a atteint sa valeur finale.

Pour compteur **allant de** initiale **à** finale **par pas** valeur du pas
Debut_pour
instructions
Fin_Pour

- Répéter…Tantque :

Répéter Debut_Répéter
Instructions
Fin_Répéter
Tantque (Condition)

On y répète des instructions jusqu'à ce qu'une certaine condition soit réalisée

Les instructions entre Répéter et Tantque sont exécutées au moins une fois et leur exécution est répétée Tantque condition soit vrai.

Choix d'un type de boucle

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus naturel d'utiliser la boucle Pour.
- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des boucles Tantque ou Répéter …Tantque

Pour le choix entre Tantque et Répéter :

- Si on doit tester la condition de contrôle avant de commencer les instructions de la boucle, on utilisera Tantque
- Si la valeur de la condition de contrôle dépend d'une première exécution des instructions de la boucle, on utilisera Répéter… tantque

Chapitre 5 : Introduction au langage C

Un programme est une série d'actions élémentaires compréhensibles par le "cerveau" de la machine, cette succession permettant de réaliser une action plus compliquée.
La machine a son propre langage appelé langage machine. Il serait trop compliqué d'écrire directement les programmes en langage dit de bas niveau. Nous utilisons donc des langages dits "évolués" compréhensibles pour un initié. Ce langage sera ensuite traduit en langage machine.
Pourquoi le langage C?
Le C est petit (32 mot clé).
Le C est populaire (Trop de programme C existent).
Le C est stable (Le langage ne change pas beaucoup).
Le C est rapide en exécution.
Le C est à la base de beaucoup d'autre langages (Java, C++).
Le C est l'un des langages faciles à apprendre.

Quand on programme en utilisant le langage C, quelques notions sont très utilisées lors de la programmation :

- *Code source*: Le programme que vous allez écrire.
- *Compilateur*: vous aidez à corriger les erreurs syntaxiques (oublier un point-virgule …) et logiques (division par zéro …) du code source et le convertit en un langage proche du langage machine.
- *Exécutable*: Le programme compilé que l'ordinateur peut exécuter.
- *Langage*: intermédiaire entre le programmeur et la machine.
- *Bibliothèque* : fichier qui contient les fonctions dédiées pour des taches bien définis (math, texte ...).
- *Fichier entête*: Fichiers se terminant par .h qui sont au début du code source.

Un programme est donc une suite d'instructions exécutées par la machine.
Ces instructions peuvent

- soit s'enchaîner les unes après les autres, on parle alors de séquence d'instructions;
- ou bien s'exécuter dans certains cas et pas dans d'autres, on parle alors de structure alternative;
- ou se répéter plusieurs fois, on parle alors de structure répétitive.

Chapitre 6: Les fonctions

Dans tout langage de programmation évolué, il est possible de définir un bloc d'instructions qui pourra être appelé dans n'importe quelle partie du programme principal en faisant référence uniquement par son identifiant. Cette technique de programmation simplifie grandement les algorithmes (programmes) et fait appel à des sous-programmes effectuant chacun des tâches précises.

Il existe deux types de sous-programme: les fonctions & les procédures

Certains problèmes conduisent à des programmes longs, difficiles à écrire et à comprendre. On les découpe en des parties appelées sous-programme ou module. Les fonctions et les procédures sont des modules (ensemble d'instructions) indépendants désigné par un nom et qui possèdent plusieurs intérêts :

- ✓ Permettent de factoriser les programmes c'est-à-dire de mettre en commun les parties qui se répètent.
- ✓ Permettent une structuration et une meilleure lisibilité des programmes.
- ✓ Facilitent la maintenance du code (il suffit de modifier une seule fois).
- ✓ Ces procédures et fonctions peuvent éventuellement être réutilisées dans d'autre programme.

6.1. Définition

Une fonction est un ensemble d'instructions exécutant une ou plusieurs tâches. La fonction est identifiée à un type et restitue une valeur en fin d'exécution. Donc, Une fonction en C est assez proche de la notion mathématique de fonction qui retourne un résultat à partir des valeurs des paramètres.

Exemples : y = sqrt(x) ;

6.2. Déclaration d'une fonction

La déclaration d'une fonction se fait, en précisant:

- Le type de valeur renvoyée par la fonction
- Le nom de la fonction
- Les types d'arguments

La ligne de déclaration d'une procédure s'écrit sous la forme:

```
Type_de_donnee   Nom_De_La_Fonction(type_argument1,   type_argument2, ...);
```

Une fonction possède trois aspects:

1. Le prototype : c'est la déclaration nécessaire avant tout.
2. L'appel : c'est l'utilisation d'une fonction à l'intérieur d'une autre fonction (par exemple le programme principale).
3. La **définition** : c'est l'écriture proprement dite de la fonction, en-tête et corps.

6.3. Les procédures

Une procédure peut être définit comme étant un ensemble d'instructions, identifié par un nom, qui exécutent une ou plusieurs tâches. Elle est un bloc de programme qui travaille sur des données fournies ou contenues dans le bloc de programme. Aucune valeur n'est associée au nom de la procédure.

Remarque : Une procédure est une fonction qui ne renvoie aucune valeur.

La déclaration d'une procédure se fait de la même manière qu'une fonction avec la différence de ne pas préciser son type.
La ligne de déclaration d'une procédure s'écrit sous la forme:

```
Void Nom_De_La_Fonction(type_argument1, type_argument2, ...);
```

Chapitre 7: les fonctions récursif

7.1. Définition

La récursivité est une méthode de description d'algorithmes qui permet à une procédure (ou une fonction) de s'appeler elle-même. La fonction fct() ci-dessous s'appelle elle-même :

```
void fct () {
      ……
     fct() ;
}
```

La forme récursive permet généralement l'écriture des fonctions sous une forme concise et plus simple à comprendre. Toutefois, elle peut être moins naturelle à concevoir. Lorsque le problème traité peut se décomposer en une succession de sous-problèmes identiques, la récursivité est généralement bien indiquée.
Pour la forme récursive, nous allons nous appuyer sur une autre écriture de la factorielle :

$$!n = n \times !(n-1)$$

Cette écriture permet l'introduction de la récursivité car elle fait intervenir la factorielle (d'où la récursivité). Voici l'implémentation de la fonction récursive en C :

```
int factorielle (int N) {
  if (N<=1) return 1;          // Si N <= 1, retourne 1 car !0=1 et
!1=1
  return N*Factorielle(N-1);  // Retourne N*!(N-1) }
```

La forme récursive est généralement plus simple à comprendre et plus élégante, elle peut être séduisante dans sa conception intellectuelle. Mais les appels récursifs occasionnent la sauvegarde du contexte (les valeurs des variables) avant chaque appel et sa restitution au retour de l'appel, ce qui peut légèrement diminuer l'efficacité du programme.

Chapitre 8 : Concept d'algorithme récursif

Les exemples d'utilisation des fonctions récursives que nous avons vus jusqu'à présent avaient tous une nature récursive, car ils mettaient en œuvre des éléments imbriqués les uns dans les autres.

Comme nous allons le voir, il aurait tout à fait été possible de programmer ces exemples sans utiliser de fonctions récursives. De la même manière, il n'est pas nécessaire qu'un problème ait en lui-même une nature récursive, pour qu'il soit possible de le résoudre très simplement avec une fonction récursive.

Prenons par exemple le calcul de la factorielle d'un nombre, une fonction mathématique qui pour une valeur entière positive, retourne le produit de tous les entiers entre 1 et cette valeur. Pour une valeur nulle, la fonction retourne 1. Par exemple, la factorielle de 5, que l'on note "5!", vaut 1*2*3*4*5=120. On peut écrire la fonction factorielle sous la forme d'une simple boucle, de la manière suivante :

```c
#include <stdio.h>
unsigned int n, i, result;

unsigned int factoriel(unsigned int n);

unsigned int factoriel(unsigned int n){
unsigned int f = 1;
for (i = 1; i <= n; i++)
 f = f * i;
return f;
}
int main(){
 printf("Entrer un entier positif :");
 scanf("%d", &n);
 result = factoriel(n);
 printf("Le factoriel de %d est %d\n", n, result);
 return 0;
}
```

Il est cependant possible de donner une définition récursive de la fonction factorielle. La factorielle d'un nombre N vaut 1 si N est égal à 0, et N multiplié par la factorielle de N - 1 sinon. Cette définition est parfaitement équivalente à la précédente, et peut se traduire en code par une fonction récursive (voir 3.1).

8.1. Avantages et inconvénients

Une grande partie des problèmes peut se résoudre avec une implémentation récursive, comme avec une implémentation itérative. L'une ou l'autre peut paraître plus ou moins naturelle suivant le problème, ou suivant les habitudes du programmeur. Avec un peu d'habitude, utiliser l'implémentation récursive permet souvent d'avoir un programme plus simple, plus facile à comprendre, donc à débugger.

L'implémentation récursive a cependant deux principaux inconvénients, qui peuvent être gênants dans certains cas :

- ✓ Un appel de fonction prend plus de temps qu'une simple itération de boucle.
- ✓ Un appel de fonction utilise une grande quantité de mémoire.

Le premier inconvénient fait que des programmes implémentés avec une fonction récursive seront souvent légèrement plus lents que leurs équivalents itératifs. Si le moindre gain de vitesse pour cette partie de votre programme est important, il peut donc être préférable d'utiliser une implémentation itérative. Dans le cas contraire, la perte de performances peut être largement compensée par le gain en clarté du code, donc en réduction de risques de laisser des bugs.

Le deuxième inconvénient peut être très gênant si le nombre d'appels imbriqués est très important. Chaque appel de fonction imbriqué utilise une certaine quantité de mémoire, plus ou moins importante selon le nombre de paramètres et de variables de votre fonction. Cette mémoire est libérée dès que l'exécution de la fonction se termine, mais dans le cas d'une fonction récursive, cette quantité de mémoire est multipliée par le nombre d'appels imbriqués à un moment donné. Si ce nombre d'appels imbriqués peut atteindre des centaines de milliers, voire des millions, on peut facilement atteindre des méga-octets de mémoire, pour un calcul qui ne prendrait aucune mémoire avec une fonction itérative.

Dans certains cas, le compilateur est capable d'éviter de lui-même ces deux inconvénients, en transformant automatiquement votre fonction récursive en un programme itératif. Ceci reste cependant assez rare, et il ne faut donc pas trop compter dessus avec les compilateurs actuels.

8.2. Passage du récursif à l'itératif

Un programme itératif se base sur des boucles pour traiter un certain nombre d'éléments. Un programme itératif simple peut donc ressembler à l'exemple suivant, qui affiche un certain nombre de fois un caractère :

```
void afficheLigne(int nb, char c){
 int i;
 for(i = 0; i < nb; i++)
  printf("%c", c);
 printf("\n");
}
```

Pour écrire une version récursive de ce programme, on commence par se demander dans quel cas la boucle n'est pas du tout utilisée. En l'occurrence, il s'agit du cas où le paramètre nb vaut 0, donc qu'on ne fait qu'afficher le retour à la ligne. On peut alors commencer à écrire une fonction qui gère ce cas :

```
void afficheLigne(int nb, char c){
 if(nb == 0)
  printf("\n");
}
```

Reste à gérer le cas où il y a des choses à afficher. Le principe de la fonction récursive est qu'elle s'occupe d'une seule étape, et laisse les étapes suivantes pour les appels imbriqués. Dans le cas où il y a des caractères à afficher, la fonction doit donc afficher un caractère, puis se

rappeler, avec comme paramètre le nombre de caractères restant à afficher. Il s'agit de la valeur qu'on lui a transmise, diminuée de 1 :

```
void afficheLigne(int nb, char c){
 if(nb == 0)
  printf("\n");
 else{
  printf("%c", c);
  afficheLigne(nb-1, c);
 }
}
```

Cette fonction réalise exactement la même chose que la version itérative. On peut ainsi dire en français : pour afficher une ligne de N caractères, il faut afficher un caractère, puis afficher une ligne de N-1 caractères.

Chapitre 9 : les Pointeurs et allocation dynamique de la mémoire

9.1. Définition

Un pointeur est aussi une variable, il est destiné à contenir une adresse mémoire, c'est à dire une valeur identifiant un emplacement en mémoire. Pour différencier un pointeur d'une variable ordinaire, on fait précéder son nom du signe '*' lors de sa déclaration.

Déclaration :

La syntaxe de la déclaration d'un pointeur est : type *variable. Exemple: int *x (x est un pointeur de type entier).

Un pointeur contient en principe une **adresse valide**, donc il peut prendre l'adresse d'une variable existante, reprenons l'exemple 1:

**int ***px; Réserve un emplacement pour stocker une adresse mémoire.

px = &x; Ecrit l'adresse de x dans cet emplacement.

Le pointeur **px** ayant comme valeur cette adresse, nous pouvons donc utiliser ce pointeur pour aller chercher (lire ou écrire) la valeur de x. Pour cela on fait précéder le nom du pointeur de l'opérateur de déréférencement '*'. Donc l'instruction suivante :

*printf("%d",*px);*

On peut modifier la valeur de x : *px = 20; Maintenant x est égal à 20.

***Remarque :* Le pointeur NULL**

Il s'agit d'un pointeur ne pointant sur rien. Elle est utilisée pour des raisons de lisibilité et de sécurité.

9.2. Pointeurs et tableaux

Prenons le cas des tableaux : Le nom d'un tableau sans décoration retourne l'adresse du premier élément du tableau. Ceci fait que l'on peut l'utiliser de la même manière qu'un pointeur. Soit le tableau Tab suivant :

int Tab[10]={5,8,4,3,9,6,5,4,3,8};

L'instruction suivante affiche bien la valeur du premier élément du tableau par pointeur déréférencé.

printf("%d",*Tab);

Ceci nous montre que le nom d'un tableau peut s'utiliser comme un pointeur sur son premier élément. On peut alors déclarer un pointeur et l'initialiser avec le nom du tableau. A condition bien sûr qu'il soit du même type, pointeur sur des entiers dans notre cas :

**int ***pTab;
pTab = Tab;

On peut donc se servir de ce pointeur comme s'il était un tableau et des opérateurs crochets [] pour accéder à ses éléments :

printf("%d",pTab[0]); ou printf("%d",*pTab); // Affiche 5.

Si on incrémente le pointeur pTab il ne contiendra pas l'adresse du premier élément du tableau + 1, mais l'adresse de l'élément suivant. La valeur de l'adresse qu'il contient sera donc incrémentée de la taille du type qu'il référence. Ceci est l'une des raisons pour lesquelles il faut donner un type à un pointeur. Donc si nous écrivons :

printf("%d",*(Tab + 1)); //Affiche 8.

9.3. Pointeurs et chaînes de caractères

On peut attribuer l'adresse d'une chaîne de caractères constante à un pointeur sur **char.**
Exemple :

```
char  *C;
C  =  "Hello world";
```

Un pointeur sur char peut être initialisé lors de la déclaration si on lui affecte l'adresse d'une chaîne de caractères constante :

```
char  *B  =  "Bonjour !" ;
```

Attention : Il existe une différence importante entre les deux déclarations suivantes :

```
char  A[]  =  "Bonjour !";    /*  un  tableau  */
char  *B   =  "Bonjour !";    /*  un  pointeur  */
```

A est un tableau qui a exactement la grandeur pour contenir la chaîne de caractères et la terminaison **\ 0**. Les caractères de la chaîne peuvent être changés, mais le nom **A** va toujours pointer sur la même adresse en mémoire.

B est un pointeur qui est initialisé de façon à ce qu'il pointe sur une chaîne de caractères constante stockée quelque part en mémoire. Le pointeur peut être modifié et pointer sur autre chose. La chaîne constante peut être lue, copiée ou affichée, mais pas modifiée.

9.4. Allocation dynamique de la mémoire

Déclaration statique de données chaque variable dans un programme a besoin d'un certain nombre d'octets en mémoire. Jusqu'ici, la réservation de la mémoire s'est déroulée automatiquement par l'emploi des déclarations des données. Dans tous ces cas, le nombre d'octets à réserver était déjà connu pendant la compilation. Nous parlons alors de la déclaration statique des variables.

Exemple :

```
float A, B, C; /* réservation de 12 octets */
short D[10][20]; /* réservation de 400 octets */
char E[] = {"Bonjour !"}; /* réservation de 10 octets */
char F[][10] = {"un", "deux", "trois", "quatre"}; /* réservation
de 40 octets */
```

Nous avons vu que l'utilisation de pointeurs nous permet de mémoriser économiquement des données de différentes grandeurs. Si nous générons ces données pendant l'exécution du programme, il nous faut des moyens pour réserver et libérer de la mémoire au fur et à mesure que nous en avons besoin.

9.4.1. Problème

Nous avons vu que l'utilisation de tableaux nous force à réserver plus de places en mémoire que ce qu'il en faut. Si nous générons ces données, dont nous ne pouvons pas prévoir le nombre et la taille lors de la programmation, il nous faut, en plus de l'utilisation des pointeurs, des moyens pour réserver et libérer de la mémoire au fur et à mesure que nous en avons besoin. Ce serait alors un gaspillage de réserver toujours l'espace maximal prévisible. Il nous faut donc un moyen de gérer la mémoire lors de l'exécution du programme. Nous parlons alors de l'allocation dynamique de la mémoire.

9.4.2. Définition

L'allocation dynamique de mémoire permet la réservation d'un espace mémoire pour son programme au moment de son exécution. Ceci est à mettre en opposition avec l'allocation statique de mémoire. En effet, dans ce type d'allocation, la mémoire est réservée dès le début de l'exécution d'un bloc.

9.4.3. La fonction malloc () et l'opérateur sizeof

La fonction malloc() de la bibliothèque <stdlib> nous aide à localiser et à réserver de la mémoire au cours d'un programme. La fonction malloc() a la signature suivante :

void* malloc (size_t taille);

Cette signature montre tout d'abord que le résultat fourni par malloc() est un "pointeur générique". Il pourra donc être converti implicitement en un pointeur de n'importe quel type.

D'autre part, nous constatons que l'unique argument de malloc est d'un type a priori inattendu (nous aurions pu penser à int ou long). En fait, size_t est un nom de type prédéfini.

La fonction prend un paramètre : le nombre d'octets à réserver. Ainsi, il suffira d'écrire sizeof(int) dans ce paramètre pour réserver suffisamment d'espace pour stocker un **int**. Notons que l'inclusion du **stdlib.h** est obligatoire.

9.4.4. La fonction free()

L'un des intérêts essentiels de la gestion dynamique de la mémoire est de pouvoir récupérer des emplacements dont on n'a plus besoin. Le rôle de la fonction free est de libérer un emplacement préalablement alloué. La fonction free () a la signature suivante :

void free(void* pointeur);

9.4.5. La fonction realloc()

Une autre fonction pour l'allocation dynamique de la mémoire est realloc(), cette dernière possède la signature suivante :

void * realloc (void * pointeur, size_t taille);

Cette fonction permet de modifier la taille d'une zone préalablement allouée (par malloc, calloc ou realloc). Le pointeur doit être l'adresse de début de la zone dont on veut modifier la taille.
Quant à taille de type size_t, elle représente la nouvelle taille souhaitée. Si une erreur se produit, la fonction retourne le pointeur NULL. Si en revanche, tout s'est bien passé, la fonction renvoie un pointeur vers l'adresse du nouveau bloc réalloué.

Chapitre 10: les structures de données complexes et les fichiers

10.1. Introduction

Nous avons déjà vu comment le tableau permettait de désigner sous un seul nom un ensemble de valeurs de même type, chacune d'entre elles étant repérée par un indice.
La structure, quant à elle, va nous permettre de désigner sous un seul nom un ensemble de valeurs pouvant être de types différents. L'accès à chaque élément de la structure (nommé champ) se fera, cette fois, non plus par une indication de position, mais par son nom au sein de la structure.

10.2. Déclaration d'une structure

Soit la déclaration de structure suivante :

```
struct prod{
  int numero;
  int qte;
  float prix; };
```

Celle-ci définit un modèle de structure mais ne réserve pas de variables correspondantes à cette structure. Ce modèle s'appelle ici **prod** et il précise le nom et le type de chacun des champs constituant la structure (numero, qte et prix).
Une fois un tel modèle défini, nous pouvons déclarer des variables du type correspondant (souvent, nous parlerons de structure pour désigner une variable dont le type est un modèle de structure). Par exemple :

struct prod art1;

10.3. Utilisation d'une structure

En C, on peut utiliser une structure de deux manières :
● En travaillant individuellement sur chacun de ses champs ;
● En travaillant de manière globale sur l'ensemble de la structure.
Chaque champ d'une structure peut être manipulé comme n'importe quelle variable du type correspondant. La désignation d'un champ se note en faisant suivre le nom de la variable structure de l'opérateur « point » (**.**) suivi du nom de champ tel qu'il a été défini dans le modèle. Voici quelques exemples utilisant le modèle structure et les variables art1 et art2 déclarées de ce type.

art1.numero = 15 ;

Affecte la valeur 15 au champ numero de la structure art1.

printf ("%e", art1.prix);

Affiche, suivant le code format %e, la valeur du champ prix de la structure art1.

scanf ("%e", &art2.prix);

Lit, suivant le code format %e, une valeur qui sera affectée au champ prix de la structure art2. Notez bien la présence de l'opérateur &.

art1.numero++

Incrémente de 1 la valeur du champ numero de la structure art1.

- ***Structure comportant de tableaux***

Soit la déclaration suivante :

```
 struct personne{
  char nom[30];
  char prenom [20];
  float heures [31];
} employe, courant;
```

Celle-ci réserve les emplacements pour deux variables nommées employe et courant. Ces dernières comportent trois champs :

nom qui est un tableau de 30 caractères ;
prenom qui est un tableau de 20 caractères ;
heures qui est un tableau de 31 flottants.

Voici un exemple d'initialisation d'une structure de type personne lors de sa déclaration : **struct** personne employe={"Abou", "Sara", {8,7,8,6,8,0,0,8}};

10.4. Structure de données linéaires

Parmi les structures de données linéaires il y a :

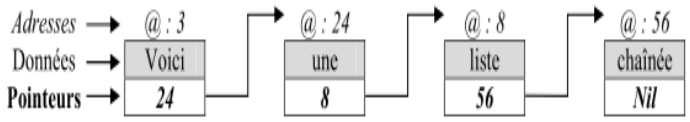
- ✓ Les tableaux,
- ✓ Les listes chaînées,
- ✓ Les piles,
- ✓ Les files.

Les structures de données linéaires induisent une notion de séquence entre les éléments les composant (1er , 2ème , 3ème , suivant, dernier…).

10.4.1. Les listes chaînées :

Une liste chaînée est une structure linéaire qui n'a pas de dimension fixée à sa création. Ses éléments de même type sont éparpillés dans la mémoire et reliés entre eux par des pointeurs. Sa dimension peut être modifiée selon la place disponible en mémoire. La liste est accessible uniquement par sa tête de liste c'est-à-dire son premier élément.

Pour les listes chaînées la séquence est mise en œuvre par le pointeur porté par chaque élément qui indique l'emplacement de l'élément suivant. Le dernier élément de la liste ne pointe sur rien (Nil).
Soit la liste chaînée suivante :



- Pour accéder au troisième élément de la liste il faut toujours débuter la lecture de la liste par son premier élément dans le pointeur duquel est indiqué la position du deuxième élément. Dans le pointeur du deuxième élément de la liste on trouve la position du troisième élément… Pour ajouter, supprimer ou déplacer un élément il suffit d'allouer une place en mémoire et de mettre à jour les pointeurs des éléments.
- Pour déclarer une liste chaînée, il suffit de créer le pointeur qui va pointer sur le premier élément de votre liste chaînée, aucune taille n'est donc à spécifier.

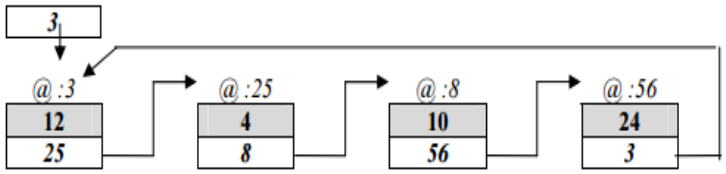
10.4.1.1. Les listes doublement chaînées :

Il existe aussi des liste chaînées, dites bidirectionnelles, qui peuvent être parcourues dans les deux sens, du 1er élément au dernier et inversement. Une liste chaînée bidirectionnelle est composée :

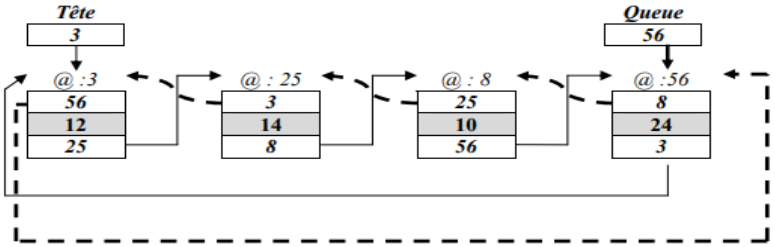
- d'un ensemble de données,
- de l'ensemble des adresses des éléments de la liste,
- d'un ensemble de pointeurs Suivant associés chacun à un élément et qui contient l'adresse de l'élément suivant dans la liste,
- d'un ensemble de pointeurs Precedent associés chacun à un élément et qui contient l'adresse de l'élément précédent dans la liste,
- du pointeur sur le premier élément Tête, et du pointeur sur le dernier élément, Queue

10.4.1.2. Les listes circulaires

Une liste chaînée peut être circulaire, c'est à dire que le pointeur du dernier élément contient l'adresse du premier. Ci-dessous l'exemple d'une liste simplement chaînée circulaire : le dernier élément pointe sur le premier.



Puis l'exemple d'une liste doublement chaînée circulaire. : Le dernier élément pointe sur le premier, et le premier élément pointe sur le dernier.



10.4.2. Les Piles

Une pile est gérée suivant la politique LIFO (Last In First Out) (dernier arrivé premier servi), ce qui signifie en clair que les derniers éléments à être ajoutés à la pile seront les premiers à être récupérés.

Une pile est gérée suivant la politique LIFO (Last In First Out) (dernier arrivé premier servi), ce qui signifie en clair que les derniers éléments à être ajoutés à la pile seront les premiers à être récupérés.

Les opérations d'insertions et des suppressions se font à une seule extrémité appelée sommet de pile.

10.4.2.1. Représentation des piles

- *Représentation contiguë*

Il existe plusieurs manières pour représenter une Pile statique. Dans cette représentation, les éléments de la pile sont rangés dans un tableau. De plus, il faut conserver l'indice du sommet de la pile et la taille maximale du tableau utilisé. Une pile statique est un enregistrement à 2 cases :

• un tableau à hauteur maximale prévisible (généralement les éléments sont des structures de données : étudiant, etc...)

• un indice entier qui pointe la dernière valeur ajoutée à la pile (sommet).

```
struct pile{
int Tab[100];
int sommet;
 };
typedef struct pile mapile;
```

- *Représentation chaînée*

Du point de vue implémentation, il n'y a aucune différence entre une Pile dynamique et une liste chainée. La seule différence réside dans les opérations qu'on effectue sur la pile. Sur une Pile, l'ajout et la suppression d'un élément ne doivent s'opérer qu'au début de la pile (Tête)

Voici donc la structure qui constituera notre pile :

```
 struct pile{
 int Val;
 struct  pile *precedent;
 };
typedef struct pile mapil;
mapil *tete=NULL;
```

10.4.3. Les files

10.4.3.1. Définition

Une file, ou file d'attente, est une liste chaînée d'informations dans laquelle : Un élément ne peut être ajouté qu'à la queue de la file, Un élément ne peut être retiré qu'à la tête de la file. Il s'agit donc d'une structure de type **FIFO** (First In First Out). Les données sont retirées dans l'ordre où

elles ont été ajoutées. Une file est comparable à une queue de clients à la caisse d'un magasin. Les files servent à traiter les données dans l'ordre où on les a reçues et permettent de :

- gérer des processus en attente d'une ressource système (par exemple la liste des travaux à éditer sur une imprimante)
- construire des systèmes de réservation.

Pour ne pas avoir à parcourir toute la liste au moment d'ajouter un élément en queue, on maintient un pointeur de queue. Attention une file peut très bien être simplement chaînée même s'il y a un pointeur de queue.

10.4.3.2. Représentation des files

- *Représentation contiguë*

Une file statique est un enregistrement à 3 cases : • Un tableau à hauteur maximale prévisible. • Un indice entier qui pointe le premier élément insérer dans la file (Début). • Un deuxième indice entier qui pointe la dernière valeur ajoutée (Queue).

```
struct file{
        int tab[10];
        int debut;
        int Queue; };
```

- *Représentation chaînée*

L'intérêt de représenter une file avec une liste chaînée est le même que pour les piles : la taille ne sera pas limitée a priori. De plus, on n'a pas besoin d'utiliser d'implémentation circulaire (à la différence d'une implémentation par tableau). Cependant, on aura aussi besoin de manipuler à la fois le début (pour défiler) et la fin (pour enfiler) de la liste, donc nous utiliserons une liste doublement chaînée. Arbitrairement, on décide de placer la tête au début de la liste. Voici donc la structure qui constituera notre file:

```
struct case{
 int info;
 case * suivant;
};
```

```
struct file{
 case *debut;
 case *Queue; };
```

10.5. Les Fichiers

10.5.1. Introduction

Heureusement, on peut lire et écrire dans des fichiers en langage C. Ces fichiers seront écrits sur le disque dur de votre ordinateur : l'avantage est donc qu'ils restent là, même si vous arrêtez le programme ou l'ordinateur.

Pour lire et écrire dans des fichiers, nous allons avoir besoin de réutiliser tout ce que nous avons appris jusqu'ici : pointeurs, structures, chaînes de caractères, etc

La valeur de mode détermine le mode d'ouverture du fichier :

r : Ouvre le fichier en lecture. Le pointeur de flux est placé au début du fichier.

r+ : Ouvre le fichier en lecture et écriture. Le pointeur de flux est placé au début du fichier.

w : Ouvre le fichier en écriture. Le pointeur de flux est placé au début du fichier.

w+ : Ouvre le fichier en lecture et écriture. Le fichier est créé s'il n'existait pas. S'il existait déjà, sa longueur est ramenée à 0. Le pointeur de flux est placé au début du fichier.

a : Ouvre le fichier en ajout (écriture à la fin du fichier). Le fichier est créé s'il n'existait pas. Le pointeur de flux est placé à la fin du fichier.

a+ : Ouvre le fichier en lecture et ajout (écriture en fin de fichier). Le fichier est créé s'il n'existait pas. La tête de lecture initiale du fichier est placée au début du fichier mais la sortie est toujours ajoutée à la fin du fichier.

10.5.2. Ouverture du fichier ***fopen***

FILE fopen(const char* nomDuFichier, const char* modeOuverture);*

Cette fonction attend deux paramètres :
Le nom du fichier à ouvrir ;
Le mode d'ouverture du fichier, c'est-à-dire une indication qui mentionne ce que vous voulez faire : seulement écrire dans le fichier, seulement le lire, ou les deux à la fois.
Cette fonction renvoi un pointeur sur FILE, C'est un pointeur sur une structure de type FILE. Cette structure est définie dans *stdio.h*.

10.5.3. Fermeture du fichier ***fclose***

Si l'ouverture du fichier a réussi, vous pouvez le lire et y écrire. Une fois que vous aurez fini de travailler avec le fichier, il faudra le « fermer ». On utilise pour cela la fonction *fclose* qui a pour rôle de libérer la mémoire, c'est-à-dire supprimer votre fichier chargé dans la mémoire vive.

Son prototype est : int fclose (FILE* pointeurSurFichier);

10.5.4. Ecriture dans un fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Ce sera à vous de choisir celle qui est la plus adaptée à votre cas.
Voici les trois fonctions que nous allons étudier :

fputc: écrit un caractère dans le fichier (UN SEUL caractère à la fois) ;
fputs: écrit une chaîne dans le fichier ;
fprintf: écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
void main()  {
double a=1.5, b=2.5;
 FILE *fichier;
// Ouverture du fichier en écriture grâce à "w" fichier
fichier = fopen( essai.tx "essai.txt , w ); ","w");
// Verifier que le fichier a bien été ouvert
if (fichier != NULL) { //Ecriture
 fprintf(fichier,"%lf\n" ,a) ;
 fprintf(fichier,"%lf\n",b);
//Fermeture du fichier
fclose (fichier); } }
```

10.5.5. Lecture à partir d'un fichier

Nous pouvons utiliser quasiment les mêmes fonctions que pour :

fgetc: lit un caractère ;
fgets: lit une chaîne ;
fscanf: lit une chaîne formatée.

Exemple :

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int i;
double tab[2];
FILE *fichier; //Ouverture du fichier en lecture grâce à "r"
fichier = fopen("essai.txt","r");
if(fichier != NULL) {
   for(i=0;i<2;i++) fscanf (fichier, "%lf\n",tab+i);
   fclose(fichier);}
for(i=0;i<2;i++) printf ("%lf\n",tab[i]);
return 0;}
```

10.5.6. Se déplacer dans un fichier

Chaque fois que vous ouvrez un fichier, il existe en effet un curseur qui indique votre position dans le fichier. Vous pouvez imaginer que c'est exactement comme le curseur de votre éditeur de texte (tel Bloc-Notes). Il indique où vous êtes dans le fichier, et donc où vous allez écrire.

En résumé, le système de curseur vous permet d'aller lire et écrire à une position précise dans le fichier.
Il existe trois fonctions à connaître :
ftell: indique à quelle position vous êtes actuellement dans le fichier ;
fseek: positionne le curseur à un endroit précis ;
rewind: remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

10.5.7. Le fichier binaire

On a 2 types de fichiers : fichiers « texte» et « binaire ».

• Le format binaire va occuper beaucoup moins de place en mémoire

• Comme il s'agit d'une recopie directe de la mémoire, on n'a pas à savoir comment est codé chaque nombre. De même que pour les fichiers texte, pour pouvoir utiliser les fichiers binaires, on doit inclure la bibliothèque d'entrées-sorties :

#include<stdio.h>

Modes d'ouverture de fichiers binaires : - "rb" ( ) read : lecture - "wb" (write) : écriture (le fichier est écrasé s'il existe) - "ab" (append) : écriture à la fin d'un fichier existant 1
Après avoir utilisé un fichier, il faut le refermer en utilisant ***fclose.*** La fonction fclose prend en paramètre le pointeur de fichier et ferme le fichier.
Ecriture d'un bloc de données en binaire

int fwrite (void *source, int taille_type, int nombre, FILE *flot)

- Ecrit tout un bloc de données en un seul appel

- Retourne un entier = nombre d'éléments effectivement écrits

Lecture d'un bloc de données en binaire
int fread(void *destination, int taille_type, int nombre, FILE *flot)
fread : - Lit un bloc de données en un seul appel - Retourne un entier = nombre d'éléments effectivement lus.
Exemple :

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 10000
void main() {
int i; double sum,tab1[DIM],tab2[DIM];
FILE *fichier; //Remplissage du tableau
for(i=0;i<DIM ;i++)
tab[i]=i*atan(1) ;
//Ecriture du fichier au format binaire
fichier=fopen("essai.bin","wb");
if (fichier !=NULL)
{fwrite(tab1,sizeof(double),DIM,fichier) ;
fclose(fichier);}
//Lecture du fichier
fichier=fopen("essai.bin","rb") ;
if (fichier !=NULL) { fread (tab2,sizeof(double),DIM, fichier);
fclose (fichier);
}
```

Comparaison : fichiers binaires et fichiers textes

	Fichiers textes	Fichiers binaires
Taille	Peu compacte	compacte
Lisible avec un programme courant	oui	non
Lisible avec un programme spécifique	oui	oui
Écriture/Lecture par blocs	**non**	**oui**
Fonctions	fopen(mode r,w ou a) fclose() fprintf() fscanf()	fopen(mode rb, wb ou ab) fclose() fwrite() fread()

Tableau 1 Comparaison : fichiers binaires et fichiers textes

Travaux Dirigés

Informatique I : TRAVAUX DIRIGES

TD N°1 Système de numérotation et de codage d'information

Exercice 1 :

1. Convertir les nombres suivants en décimal :
   $(01001101)_2$, $(1000001)_2$, $(355)_8$, $(A1C4)_{16}$, $(125)_{16}$.
2. Convertir les nombres décimaux :
   a. 100, 42,001 en binaire.
   b. 2257 en base 8 (octal)
   c. 64 en base 16 (hexadécimal)
3. Convertir le nombre binaire $(101111010001)_2$ en octal et en hexadécimal.
4. Convertir le nombre hexadécimal F6A3 en base 2 et 8.

Exercice 2 :

Coder avec le minimum de bits possibles, les nombres décimaux suivants en trois représentations : signe et de la valeur absolue, complément à 1 et complément à 2.

+9, -5, +205, -198, +127, -127, +128, -128.

Exercice 3 :

Donnez la traduction à laquelle correspond le mot de 4 octets codé en hexadécimal suivant :

$(49\ 55\ 50\ 31)_{16}$ selon qu'on le lit comme :

- un entier signé,
- un entier représenté en complément à 2,

Exercice 4 :

Quelle est la valeur décimale en complément à 2 sur 16 bits des nombres hexadécimaux :

a/ $DCFE_{16}$ b/ 4321_{16} c/ $F00F_{16}$ d/ 1000_{16}

Exercice 5 :

Soit le nombre signés N1 codés en binaire sur 8 bits où N1= -70.

1. Donner la valeur binaire de l'entier N1 sur 8bits suivant les 3 représentations, Signe et valeur absolue, Complément à 1 et complément à 2.
2. Donner la valeur décimale et binaire sur 8 bits de l'entier signé N2 suivant les trois cas ci-dessous :
   a. (N2)2=(N1)2+(0000 0001)2 où les nombres sont représentés en SVA sur 8 bits.
   b. (N2)2=(N1)2+(0111 1111)2 où les nombres sont représentés en Complément à 1 sur 8 bits.
   c. N1-N2=5 où les nombres sont représentés en Complément à 2 sur 8 bits.

Correction TD N°1

Exercice 1 :

1. Le nombre $(01001101)_2$ en base 10 est $2^0 + 2^2 + 2^3 + 2^6 = 1+4+8+64 = 77$.

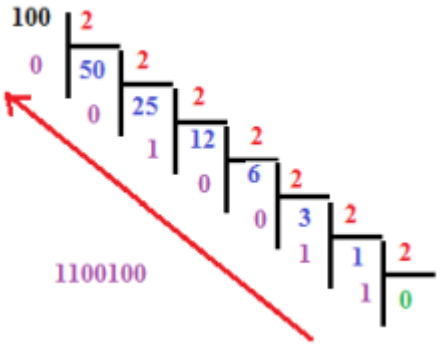
   $(1000001)_2 = 1*2^0+1*2^6=1+64=65$.

   $(355)_8 = 5*8^0+5*8^1+3*8^2=237$.

   $(A1C4)_{16} = 4*16^0+12*16^{1+}1*16^2+15*16^3 = 41412$.

   $(125)_{16} = 5*16^0+2*16^1+1*16^2=293$.

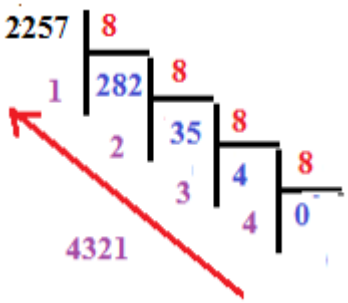
2. a) $100 = (1100100)_2$



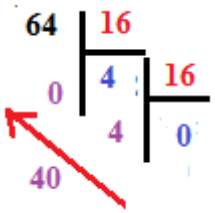
$42 = (101010)_2$

$001 = (1)_2$

b) $2257)_{10} = (4321)_8$



c) $64 = (40)_{16}$



3. $(001\ 001\ 001\ 001\ 001\ 001)_2 = (111111)_8$

$(1001\ 0010\ 0100\ 1001)_2 = (9249)16$

4. $(F6A3)_{16} = (\ 1111\ 0110\ 1010\ 0011)_2$

$= (\ 1\quad 7\quad 3\quad 2\quad 4\quad 3\ )_8$

Exercice 2 :

+9 = (+) → 01001 sva,cp1 et cp2

-5 = (-) → 1101 sva
0101
1010 cp1
\+ 1
1011 cp2

+127 = (+) → 01111111 sva,cp1 et cp2

-127 = (-) → 11111111 sva
01111111
10000000 cp1
\+ 1
10000001 cp2

+205 = (+) → 011001101 sva,cp1 et cp2

-198 = (-) → 111000110 sva
011000110
100111001 cp1
\+ 1
10011010 cp2

+128 = (+) → 010000000 sva,cp1 et cp2

-128 = (-) → 110000000 sva
010000000
101111111 cp1
\+ 1
110000000 cp2

Exercice 3 :

Hexadécimal	4	9	5	5	5	0	3	1
Binaire	0 100	1001	0 101	0101	0101	0000	0011	0001

Entier signé	+	**1 230 327 857**

Complément à 2	**+**	**1 230 327 857**

Exercice 4 :

a/ $DCFE_{16}$ = 1101 1100 1111 1110 => c'est un nombre négatif en CP à 2

\- 1

1101 1100 1111 1101 => le CP à 1

0010 0011 0000 0010 => On inverse le nombre pour calculer ça valeur

$= 2^1+2^8+2^9+2^{13} = 8962)_{10}$

=> $DCFE)_{16} =$ **- 8962**$)_{10}$

b/ 4321_{16} = 0100 0011 0010 0001 => c'est un nombre Positif en CP à 2

$=2^0+2^5+2^8+2^9+2^{14}= 17185)_{10}$

=> $4321)_{16}$ = **17185**$)_{10}$

c/ $F00F_{16}$ = 1111 0000 0000 1111 => c'est un nombre négatif en CP à 2

\- 1

1111 0000 0000 1110 => le CP à 1

0000 1111 1111 0001 => On inverse le nombre pour calculer ça valeur

$= 2^0+2^4+2^5+2^6+2^7+2^8+2^9+2^{10}+2^{11}= 4081)_{10}$

=> $F00F)_{16}$= - **4081**$)_{10}$

d/ 1000_{16} =0001 0000 0000 0000 => c'est un nombre Positif en CP à 2

$=2^{12}= 4096)_{10}$

=> $4321)_{16}$ = **4096**$)_{10}$

Exercice 5:

Soit le nombre signés**N1** codés en binaire sur 8 bits où **N1=-70**.

1°) SVA : 1100 0110 CP1 : 1011 1001 CP2 : 1011 1010

2°) A°/ $(N2)_2=(N1)_2+(0000\ 0001)_2$

⇒ 1100 0110

\+

0000 0001

N2=(1100 0111)$_2$ sur 8bits SVA→N2<0 donc N2=-71

B°/ $(N2)_2=(N1)_2+(0111\ 1111)_2$

⇒ 1011 1001

\+

0111 1111

N2=(1 0011 1000)$_2$ sur 8bits cp1→N2>0 donc N2=+56

C°/ méthode 1 :

N1-N2=5 ⇒ N2=N1-5 en CP2

⇒ 1011 1010

\-

0000 0101

N2=(1011 0101)$_2$ N2<0 cp2

(0100 1010)$_2$ +1=(0100 1011)$_2$

Donc N2= -75

méthode 2 :

N1-N2=5 ⇒ N2=N1+ (-5) en CP2

⇒ 1011 1010

\+

1111 1011 -5 en cp2

N2=(1 1011 0101)$_2$ N2<0 cp2

(0100 1010)$_2$ +1=(0100 1011)$_2$

Donc N2= -75

TD N°2 Algèbre de Boole et Circuits logiques

Exercice1 :

Soit la fonction logique suivante :

$$Y = \bar{A}BC + A\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC$$

Donner la table de vérité de Y ; A partir de la table de vérité de Y, donner la fonction$\bar{Y}$.

Exercice 2 :

Soit la table de vérité de la fonction F:

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1

1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

-Trouvez la fonction de F.

- Simplifier la fonction F (A,B,C,D) en utilisant les théorèmes de l'algèbre de Boole.

Exercice 3 :

- En utilisant les théorèmes de l'algèbre de Boole démontrez que :

$f(x,y,z) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = xy + yz + zx$ et $g(A,B) = \overline{A \oplus B} = \bar{A}\bar{B} + AB$

- Simplifier la fonction logique suivante :

$E(a,b,c,d) = abcd + \bar{a} + ab + \bar{b} + \bar{c}d + \bar{c} + \bar{d}$

Remarque : a+b+c=(a+b)+c=(a+c)+(b+c)

Exercice 4:

Donnez les fonctions simplifiées de Tableaux de Karnaugh suivants :

zw / xy	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	1	0	1
10	0	0	0	1

zw / xy	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	0	0	1	0
10	1	0	1	1

Exercice 5:

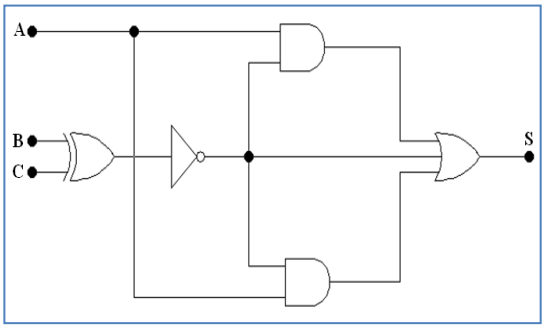
Soit les fonctions Logiques suivantes:$F(x,y,z,w) = \bar{x}\bar{y}\bar{z}w + x\bar{z}\bar{w} + xy\bar{z}w + \bar{x}yzw + \bar{y}z\bar{w} + xy\bar{z}$

$$g(x,y,z,w) = \bar{x}y\bar{z} + xyz + \bar{x}zw + x\bar{z}w$$

Simplifier au maximum les fonctions logique F et G en utilisant la méthode de Karnaugh.

Exercice 6 :

1- Donner l'expression de la fonction S (A, B, C).
2- Simplifier le circuit de la fonction S.
3- Tracer le circuit de la fonction S après la simplification.



Exercice 7

Un contrôle de qualité est effectué sur des briques dans une usine. Chaque brique possède quatre critères de qualités : - Son poids **P**, - Son épaisseur **e**, - Sa longueur **L**, - Sa largeur **l.**

Ces quatre grandeurs sont mesurées sur chaque brique. Elles sont classées en trois catégories :

- **Qualité A** : Le poids P et deux dimensions au moins sont correctes.
- **Qualité B** : Le poids seul est incorrect ou, le poids étant correct, deux dimensions au moins sont incorrectes.

1. Écrire les équations des fonctions A, B, C.
2. Simplifier ces fonctions.

Exercice 8

Quatre personne, membre de jury d'un jeu télévisé, dispose chacune d'un bouton poussoir pour permettre l'enregistrement de leur opinion concernant un disque de musique. Au lieu d'enregistrer les scores individuels, on traite l'information de façon que le panneau d'affichage indique un <succès> lorsque la majorité des votes sont favorables et un <échec> dans le cas contraire. Il faut également prévoir l'indication d'un < nul> (égalité de points).

1. Donner les expressions logiques du panneau d'affichage.
2. Simplifier les expressions logiques.

Correction TD N°2

Exercice 1:

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}BC$	$\bar{A}B\bar{C}$	ABC	Y	$\bar{Y}$
0	0	0	1	1	1	0	0	0	0	1
0	0	1	1	1	0	0	0	0	0	1
0	1	0	1	0	1	0	1	0	1	0
0	1	1	1	0	0	1	0	0	1	0
1	0	0	0	1	1	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0	1
1	1	0	0	0	1	0	0	0	0	1
1	1	1	0	0	0	0	0	1	1	0

Exercice 2 :

$F(A,B,C,D) = \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + A\bar{B}CD + ABCD + \bar{A}BCD$

$= \bar{A}B\bar{C}(\bar{D} + D) + ACD(\bar{B} + B) + \bar{A}BCD$

$= \bar{A}B\bar{C} + ACD + \bar{A}BCD \qquad (D + D) = 1$

$= \bar{A}B\bar{C} + CD(A + \bar{A}B)$

$= \bar{A}B\bar{C} + ACD + BCD$

Exercice 3 :

- $f(x,y,z) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

$= (\bar{x}yz + x\bar{y}z + xy\bar{z}) + xyz$

$= (\bar{x}yz + xyz) + (x\bar{y}z + xyz) + (xy\bar{z} + xyz)$

$= (\bar{x} + x)yz + (\bar{y} + y)xz + (\bar{z} + z)xy$

$= yz + xz + xy$

- $E(a,b,c,d) = abcd + \bar{a} + \bar{c}d + \bar{b} + ab + \bar{c} + \bar{d}$

$= abcd + \bar{a} + \bar{b} + \bar{c} + \bar{d} + \bar{c}d + ab$

$= abcd + \overline{abcd} + \bar{c}d + ab$

$= 1 + \bar{c}d + ab = 1$

Exercice 4 :

zw / xy	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	1	0	1
10	0	0	0	1

zw / xy	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	0	0	1	0
10	1	0	1	1

$F(x,y,z,w) = z\bar{w} + \bar{x}\bar{w} + xy\bar{z}w$ $F(x,y,z,w) = \bar{y}\bar{w} + \bar{x}\bar{z}\bar{w} + \bar{x}yw + xzw$

Exercice 5 :

$$F(x,y,z,w) = \bar{x}\bar{y}\bar{z}w + x\bar{z}\bar{w} + xy\bar{z}w + \bar{x}yzw + \bar{y}z\bar{w} + xy\bar{z}$$

zw / xy	00	01	11	10
00	0	1	0	1
01	0	0	1	0
11	1	1	0	0
10	1	0	0	1

$$F(x,y,z,w) = x\bar{z}\bar{w} + xy\bar{z} + \bar{y}z\bar{w} + \bar{x}\bar{y}\bar{z}w + \bar{x}yzw$$

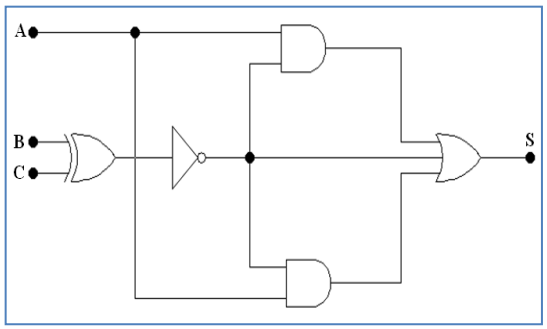
$$g(x,y,z,w) = \bar{x}y\bar{z} + xyz + \bar{x}zw + x\bar{z}w$$

zw / xy	00	01	11	10
00	0	0	1	0
01	1	1	1	0
11	0	1	1	1
10	0	1	0	0

$$g(x,y,z,w) = \bar{x}y\bar{z} + x\bar{z}w + \bar{x}zw + xyz$$

Exercice 6:

Soit le circuit suivant:



1- Déterminer l'expression logique de sa sortie S.

S (A,B,C)= $A\overline{B \oplus C} + \overline{B \oplus C} + A\overline{B \oplus C}$

2- En utilisant les lois de l'algèbre de Boole,

- Simplifier la fonction logique S.

S (A,B,C)= $A\overline{B \oplus C} + \overline{B \oplus C} + A\overline{B \oplus C} = A\overline{B \oplus C} + \overline{B \oplus C} = \overline{B \oplus C}\,(A+1) = \overline{B \oplus C}$

Exercice 7:

P	E	L	R	A	B
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0

$$\boldsymbol{A}(\boldsymbol{P},\boldsymbol{E},\boldsymbol{L},\boldsymbol{R}) = P\bar{E}LR + PE\bar{L}R + PEL\bar{R} + PELR$$

$$\boldsymbol{B}(\boldsymbol{P},\boldsymbol{E},\boldsymbol{L},\boldsymbol{R}) = \bar{P}ELR + P\bar{E}\bar{L}\bar{R} + P\bar{E}\bar{L}R + P\bar{E}L\bar{R} + PE\bar{L}\bar{R}$$

Qualité A:

PE\LR	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	1
10	0	0	1	0

$$A = PER + PEL + PLR$$

Qualité B :

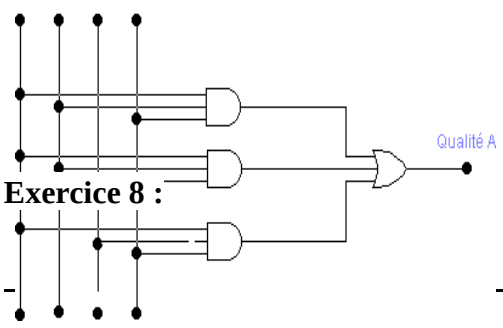
$$B = \bar{P}ELR + P\bar{E}\bar{L}\bar{R} + P\bar{E}\bar{L}R + P\bar{E}L\bar{R} + PE\bar{L}\bar{R}$$

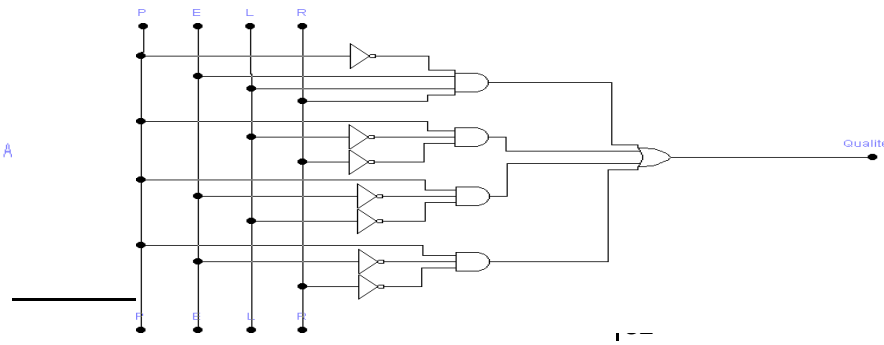
PE\LR	00	01	11	10
00		0	0	
01		0	1	
11	1		0	
10	1	1	0	1

$$B = \bar{P}ELR + P\bar{L}\bar{R} + P\bar{E}\bar{L} + P\bar{E}\bar{R}$$

Circuits logiques :

Qualité A : $\boldsymbol{A}(\boldsymbol{P},\boldsymbol{E},\boldsymbol{L},\boldsymbol{R}) = PER + PEL + PLR$ **Qualité B :** $\boldsymbol{B}(\boldsymbol{P},\boldsymbol{E},\boldsymbol{L},\boldsymbol{R}) = \bar{P}ELR + P\bar{L}\bar{R} + P\bar{E}\bar{L} + P\bar{E}\bar{R}$





Exercice 8 :

A	B	C	D	S	E	N
0	0	0	0	0	1	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	0	0	1
0	1	1	0	0	0	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	0	0

$$S=\bar{A}BC\,D+A\bar{B}C\,D+AB\bar{C}\,D+ABC\,\bar{D}+ABC\,D$$
$$E=\bar{A}\bar{B}\bar{C}\,\bar{D}+\bar{A}\bar{B}\bar{C}\,D+\bar{A}\bar{B}C\,\bar{D}+\bar{A}B\bar{C}\,\bar{D}+A\bar{B}\bar{C}\,\bar{D}$$
$$N=\bar{A}\bar{B}C\,D+\bar{A}B\bar{C}\,D+\bar{A}BC\,\bar{D}+A\bar{B}\bar{C}\,D+A\bar{B}C\,\bar{D}+AB\bar{C}\,\bar{D}$$

2. Simplification par tableau de Karnaugh:

Succès:

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	0

$$S=ABC+AC\,D+BC\,D+AB\,D$$

Échec:

AB \ CD	00	01	11	10
00	1	1	0	1
01	1	0	0	0
11	0	0	0	0
10	1	0	0	0

$$E=\bar{A}\bar{B}\bar{C}+\bar{A}\bar{C}\,\bar{D}+\bar{A}\bar{B}\bar{D}+\bar{B}\bar{C}\,\bar{D}$$

match nul:

AB \ CD	00	01	11	10
00	0	0	1	0
01	0	1	0	1
11	1	0	0	0
10	0	1	0	1

$$N=\bar{A}\bar{B}C\,D+\bar{A}B\bar{C}\,D+\bar{A}BC\,\bar{D}+A\bar{B}\bar{C}\,D+A\bar{B}C\,\bar{D}+AB\bar{C}\,\bar{D}$$

TD N°3 Algorithmique

Exercice 1 :

- Ecrire un algorithme qui affiche le message « Hello Word !!! »

Exercice 2 :

Quelles seront les valeurs des variables A et B après l'exécution des instructions suivantes ?

Variables d'entrée : A, B : **Entier ;** **Variables de sortie:** **Début** A=1; B=A+3; A=3; **Fin.**	**Variables d'entrée :** A, B : **Entier;** **Variables de sortie:** **Début** A=5; B=A+4; A=A+1; B=A–4; **Fin.**	**Variables d'entrée :** A, B, C : **Entier ;** **Variables de sortie:** **Début** A=3; B=10; C=A+B; B=A+B; A=C; **Fin.**

Exercice 3 :

Établir un algorithme qui permet d'afficher : la somme, le produit et la moyenne de 4 nombres réels.

Exercice 4:

- Ecrire un algorithme qui demande deux nombres entiers à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit pas calculer le produit des deux nombres.

Exercice 5 :

- Écrire un Algorithme qui demande à l'utilisateur un nombre N et affiche les nombres pair de 0 jusqu'à N.
- Écrire un Algorithme qui demande à l'utilisateur un nombre N et affiche les nombres premiers de 0 jusqu'à N.

Exercice 6:

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

Exercice 7 :

Ecrire un algorithme qui demande à l'utilisateur un nombre N, calculer la somme des nombres de 0 à N, puis calculer le factoriel de N.

Exercice 8 :

Donner un algorithme, qui à partir d'un prix unitaire et d'un nombre d'articles fournis en données, permettent de calculer : le prix hors taxe, la TVA et le prix TTC correspondant. Le taux de TVA sera supposé toujours égal à 17%.

Exercice 9

Ecrire un algorithme, un organigramme, qui demande à l'utilisateur la température de l'eau et afficher son état (solide, liquide, vapeur).

Exercice 10:

- Donner, un algorithme, qui permet de calculer y tel que :

$$Y = \begin{cases} X & Si\ X \leq 50 \\ \sqrt{X^3} & Si\ \ 50 < X < 100 \\ \dfrac{1}{X} & Si\ X \geq 100 \end{cases}$$

Exercice 11 :

- Ecrire un algorithme qui calcule la somme des n premiers termes de la série harmonique, c'est-à-dire la somme: 1+1/2+1/3+....+1/n.
- Ecrire un autre algorithme pour calculer la somme des **n** premiers termes de la suite suivante :

  U0=1. Un =4+2n/3n

Exercice 12 :

Écrire un algorithme qui demande à l'utilisateur de saisir 10 entiers stockés dans un tableau. Le programme doit afficher le nombre d'entiers supérieurs ou égaux à 10.

Exercice 13 :

Écrire un algorithme qui demande à l'utilisateur de saisir 10 entiers stockés dans un tableau. Le programme doit ensuite afficher l'indice du plus grand élément.

Correction TD N°2

Exercice 1 :

```
Nom : exercice 1 ;
Variables d'entrée : /
Variable intermédiaire : /;
Variables de sortie: / ;
Début
Ecrire('' hellow word !!!'' ) ;
FIN.
```

Exercice 2:

A=3 **A=6** **A=13**
B=4 **B=2** **B=13 C=13**

Exercice 3:

```
Nom : exercice 3 ;
Variables d'entrée : A, B, C, D : réels ;
Variable intermédiaire : /;
Variables de sortie: S,P, M : réels ;
Début
Lire(A) ;
Lire(B) ;
Lire(C) ;
Lire(D) ;
S=A+B+C+D ;
P=A*B*C*D ;
M=S/4;
Ecrire ("la somme=",S) ;
Ecrire ("le produit=",P) ;
Ecrire ("la moyenne=",M) ;
FIN.
```

Exercice 4 :

```
Nom exercice 4 ;
Variables d'entrées : m, n : entier ;
Variable intermédiaire : /;
Variables de sortie: / ;
Début
Ecrire ("Entrez deux nombres : ") ;
Lire ( m, n) ;
Si ( (m>0 ET n>0) OU (m<0 ET n<0)) Alors
   debut_si
      Ecrire ("Leur produit est positif") ;
```

```
   finsi
Sinon debut_sinon
          Ecrire (”Leur produit est négatif”) ; finsinon     FIN.
```

Exercice 5 :

```
Nom : exercice 5 Q1 ;
Variables d’entrée : N entier ;
Variable intermédiaire : i entier ;
Variables de sortie: N :entier ;
Début
Lire(N) ;
S=0 ;
pour (i=0 ;i<=N ;i++) faire debut_pour
     si(N%2==0) debut_si
                 ecrire(N) ;
                finsi
                           fin_pour
FIN.
```

```
Nom : exercice5 Q2 ;
Variables d’entrée : N entier ;
Variable intermédiaire : i,j,p entier ;
Variables de sortie: j entier ;
Début
Lire(N) ;
pour (j=1 ;j<=N,j++)
debut_pour
   p=0 ;
   pour (i=2 ;i<j-1 ;i++) faire debut_pour
       si(j%i==0) debut_si
                   p=1 ;
                   finsi
                              fin_pour
  si(p==0)debut_si
       ecrire(j) ;
         finsi
fin_pour
FIN
```

Exercice 6 :

```
Nom : exo6
Variables d'entrée : âge : entier ;
Variable intermédiaire : /;
Variables de sortie: / ;
Début
Lire (age)
Si (age <=6 et age >=7)  alors debut_si  ecrire ("Poussin")  fin_si
sinon debut_sinon Si (age <=9 et age >=8)  debut_si alors   ecrire
("Pupille")  fin_si
          sinon debut_sinon Si (age <=11 et age >=10)  alors
debut_si  ecrire ("Minime")  fin_si
                    sinon debut_sinon Si (age >=12)  alors debut_si
ecrire ("Minime") fin_si
                               sinon  debut_sinon ecrire ("n'est pas
mentionné") fin_sinon
fin_sinon
fin_sinon
fin_sinon
Fin
```

Exercice 7 :

```
Nom : Somme
Variables d'entrée : N : entier ;
Variable intermédiaire : i : entier;
Variables de sortie: Somme, Fact : réel;
Début
Ecrire ("Entrer la valeur de N : ")
Lire (N)
Somme = 0
Fact=1
Pour  i allant de 1 à   N  début_Pour
                 Somme = somme+i
                 Fact=Fact*i
                                        Fin_pour
               Ecrire ("la somme = ", Somme)
               Ecrire ("le Factoriel= ", Fact)
     Fin
```

Exercice 8 :

```
Nom : Exo 8 ;
Variables d'entrée : PU : réel, Qt : entier ;
Variable intermédiaire : /;
Variables de sortie: PHT,TVA, TTC ;
    Début
     Lire (PU)
     Lire (Qt)
PHT=PU*Qt
TVA= (PHT*17)/100
TTC= PHT+TVA
écrire (PHT)
écrire (TVA)
écrire (TTC)
Fin
```

Exercice 9 :

```
Nom : température_de_H2O
Variables d'entrée :  T : réel
Variable intermédiaire : /;
Variables de sortie:/
Début
Ecrire ("entrer la température de H2O : ")
Lire (T)
Si (T>100) alors debut_si
                  Ecrire ("Etat = « vapeur » ") fin_si
Sinon  debut_sinon
            Si (T<0) alors debut_si
                Ecrire ("Etat = « solide » ") fin_si
            Sinon  debut_sinon
                Ecrire ("Etat = « liquide » ") fin_sinon
       fin_sinon
Fin
```

Exercice 10 :

```
Nom : Calcule_Y
Variables d'entrée : X : entier ;
Variable intermédiaire : /;
Variables de sortie:Y : réel ;
Début
Ecrire ("Entrer la valeur de X : ")
Lire (X)
Si (X<=50) alors début_si
                  Y= X ;
                              fin_si
Sinon   début_sinon
          Si (X>=100) alors début_si
                  Y= 1/X ;
                             fin_si
         Sinon    début_sinon
               Y=sqrt (pow(X,3))
                  fin_sinon
       fin_sinon
Ecrire ("Y= ", Y)
Fin
```

Exercice 11 :

1/
```
Nom : Harmonique ;
Variables d'entrée : N Entier ;
Variable intermédiaire : i Entier ;
Variables de sortie: S réel ;
Début
Lire(N) ;
S=0 ;
pour (i=1 ;i<N ;i++) faire debut_pour
              S=S+1/i ;
                              fin_pour
Ecrire("la somme harmonique=",S) ;
FIN.
```

2/ Nom : premier_terme
Données : N : entier
Intermédiaire : i : entier
Résultats : Somme : réel
Début
Ecrire ("Entrer la valeur de N : ")
Lire (N)
Somme = 1
Pour i allant de 1 à N début_Pour
V = (4+2*i)/3*i
Somme=somme+v
Fin_pour
Ecrire ("la somme = ", Somme) Fin

Exercice 12 :

Nom : tableau
Variables d'entrée : N, T[10] : entier ;
Variable intermédiaire : i : entier ;
Variables de sortie:/
Début
Ecrire ("Entrer le nombre des éléments du tableau N : ")
Lire (N)
Pour i allant de 1 à N début_Pour
lire (T[i])
Fin_pour
Pour i allant de 1 à N début_Pour
si (T[i]>=10) alors début_si
écrire (T[i])
fin_si
Fin_pour
Fin

Exercice 13 :

Nom : tableau2
Variables d'entrée : N, indice,T[10] : entier;
Variable intermédiaire : i : entier;
Variables de sortie:/ ;
Début
Ecrire ("Entrer le nombre des éléments du tableau N : ")
Lire (N)
Pour i allant de 1 à N début_Pour
lire (T[i]) Fin_pour
indice=0
Pour i allant de 1 à N début_Pour
si (T[i]>T[indice]) alors début_si
indice=i fin_si
écrire (indice) Fin_pour Fin

Travaux Pratiques

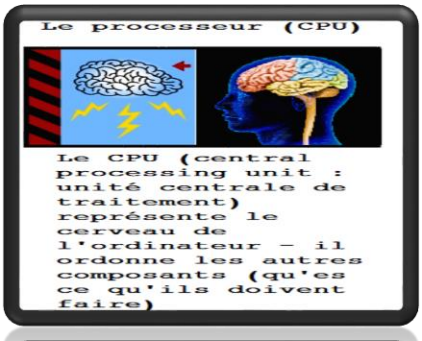
Informatique I: Travaux Pratiques

TP N°1 Architecture matérielle de l'ordinateur

Un ordinateur est une machine programmable (qu'on peut programmer). Ceci veut dire qu'il peut exécuter des instructions programmées par l'utilisateur. Aujourd'hui, le terme ordinateur fait référence à un ordinateur de bureau ou bien un ordinateur portable. L'ordinateur de bureau est généralement appelé PC (personal computer). Ce dernier contient plusieurs composants telle que la carte mère, le CPU, la mémoire (RAM), le disque dur, la carte graphique … etc. Malgré que les PC sont les ordinateurs les plus connus aujourd'hui, il existe d'autre type d'ordinateur. Par exemple, le mini-ordinateur qui est un ordinateur puissant qui peut supporter plusieurs utilisateurs à la fois. La station de travail (main frame) est un autre exemple, qui est une grande machine avec une puissance de calcul énorme qui peut exécuter des bilions de calcul à partir de plusieurs sources en même temps. Et enfin, il y a le super-ordinateur (supercomputer) qui est une machine qui peut gérer des bilions d'instructions par seconde et il est utilisé pour effectuer des calculs plus compliqués.

La carte mère est le circuit principale de l'ordinateur et elle est aussi appelée le corps logique. Si vous ouvrez l'intérieur de votre ordinateur, la grande pièce verte de silicone est la carte mère. Vous allez trouver plusieurs composants attachés à la carte mère : CPU, ROM, RAM, carte graphique, port usb … etc. Elle inclut aussi des contrôleurs de dispositifs tels que : le disque dur, le lecteur DVD, le clavier et la souris. En vraie, la carte mère est celle qui permet à tous ce qui est dans l'ordinateur de travailler en ensemble.

La RAM est l'abrégé de Random Access Memory (mémoire à accès aléatoire). Elle est composée de petites mémoires qui forment le corps de la RAM. Ces modules sont branchés dans la carte mère via ce qu'on appelle le slot mémoire. Chaque fois que vous ouvrez un programme (paint, word, jeux …) il va être chargé du disque dur vers la RAM. Ceci vint du fait que la lecture des programmes ou bien des données est plus rapide lorsqu'elle se fait à partir de la RAM qu'à partir du disque dur. Plus la taille de votre RAM est grande plus votre ordinateur pourra charger plus de données dans la RAM ce qui augmentera la vitesse de votre ordinateur.



Le CPU est l'abrégé de Central Processing Unit. Ceci est le cerveau de l'ordinateur. Il peut faire n'importe quoi allant des instructions de base (addition, soustraction …) aux fonctions complexes. Chaque fois qu'on veut calculer quelque chose, on doit l'envoyer au CPU. Chaque jours, il calcule ceci, il calcule cela –ne pas avoir peur car il n'a pas besoin de repos. Le CPU peut être appelé simplement Le processeur.

Quand vous enregistrer vos données ou bien vous installez vos programmes dans votre ordinateur, les informations sont écrites dans le **disque dur.** Le disque dur est un cylindre composé de plusieurs disques magnétiques appelés plats, qui enregistrent et stockent des informations. Puisque les données sont stockées magnétiquement dans le disque, les informations restent intactes après l'éteint de l'ordinateur. Ceci est une distinction importante entre le disque dur et la RAM qui perd tous ces informations quand l'ordinateur est éteint.

Un dispositif d'entrée est n'importe quel dispositif qui produit des données à l'ordinateur. Il existe de dizaines de dispositifs d'entrée possibles, mais les deux dispositifs les plus connues sont le clavier et la souris. Chaque bouton que vous appuyiez dans le clavier et chaque mouvement ou bien clique que vous faites avec la souris envoie un signal d'entrée à votre ordinateur. Ces entrées vous permettent d'ouvrir des programmes, d'écrire des messages, de déplacer des objets, de dessiner, de jouer, … etc. Les dispositifs d'entrée sont une partie vitale de n'importe quel ordinateur. Malgré que le clavier et la souris sont les plus connus, il existe d'autres exemples de dispositifs d'entrée : la manette de jeux, microphone, scanners, caméra digitale, webcam, lecteur de carte, … etc.

Chaque dispositif qui produit des informations de l'ordinateur est appelé **dispositif de sortie.** Puisque la plus part des sorties de l'ordinateur sont sous format graphique ou bien audio, les deux dispositifs de sortie les plus connus sont l'écran et les hautparleurs. Ces deux derniers produisent des réponses instantanées aux entrées de l'utilisateur, telles que l'affichage d'un caractère quand l'utilisateur le tape par clavier ou bien jouer un morceau de musique quand l'utilisateur le demande. Il existe d'autres exemples de dispositifs de sortie : Ecouteurs, imprimantes, projecteurs, robots, … etc.

TP N°2 Introduction au Electronic WorkBench

I. Objectif

Electronic WorkBench (EWB) est un logiciel de simulation pour les circuits électroniques. Il nous permet de concevoir et d'analyser les circuits sans utiliser des composants réels ou instruments réels.

Les circuits que nous proposons d'étudier sur l'**EWB** font partie de la famille des circuits dits "logiques". Ceux-ci sont caractérisés par le fait que leurs tensions d'entrée ou de sortie ne peuvent prendre que deux valeurs appelées niveaux logiques. La convention de logique positive définit comme suit :

- Niveau bas : absence de tension : niveau 0 ou low.
- Niveau haut : présence de tension : niveau 1 ou hight.

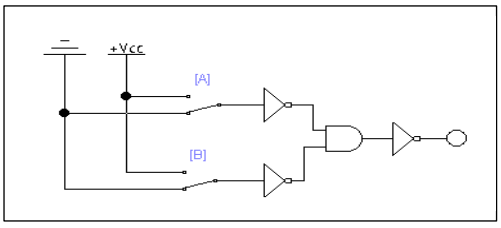
Ces circuits étant des circuits actifs, il est nécessaire d'ajouter Vcc et GND.

II. Manipulation

Dans la partie manipulation on vous demande de :

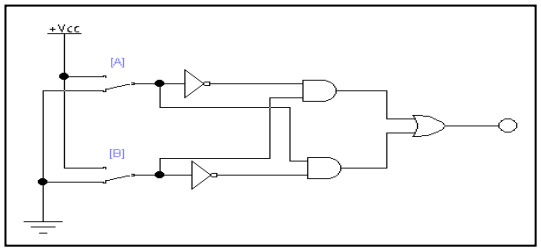
- Réaliser les circuits logiques ci-dessous.
- Vérifier que leur table de vérité correspond bien à la fonction de sortie.

Exemple 1 :



A	B	L
0	0	0
0	1	1
1	0	1
1	1	1

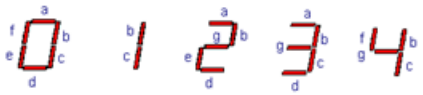
Exemple 2 :



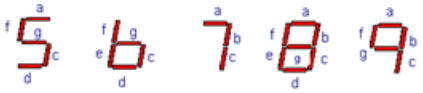
Exemple 3 : Décodeur BCD et Afficheur 7 segments

Le décodeur **BCD** (**D**écimal **C**odé **B**inaire) est destiné à l'affichage de valeurs décimales, chaque digit doit être codé en binaire sur 4 bits. Afin de simuler le fonctionnement du décodeur, nous utilisons un afficheur sept segments. Il nous permettant d'afficher des chiffres au moyen de 7 segments constitués de 7 diodes LED, c'est pourquoi nous appellerons ces afficheurs, *afficheur sept segments*.

En commandant convenablement l'allumage de certains segments, on visualise les nombres désirés.



Pour visualiser un zéro, on allumera les segments a, b, c, d, e, f. Pour visualiser un 1, on allumera les segments b, c et pour un 2, les segments a, b, g, e, d par exemple.



Pour cela nous allons procéder comme suite :

1- Compléter la table de vérité suivante :

2- En utilisant le tableau de Karnaugh, simplifier les sorties a, b, c, d, e, f, g.

3- Réaliser les circuits logiques qui forment notre décodeur.

4- Tester le décodeur pour afficher les nombres.

x1	x2	x3	x4	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

TP N°3 Manipulation de circuits logique sur EWB

I. Objectif

Exploiter les notions de la logique combinatoire pour résoudre des problèmes réels tel que le Commande Allumage et Système d'Aspiration.

II. Manipulation

Exemple 1 : Commande d'Allumage

Trois interrupteurs A, B et C commende l'allumage de deux lampes R et S suivant les conditions suivantes :

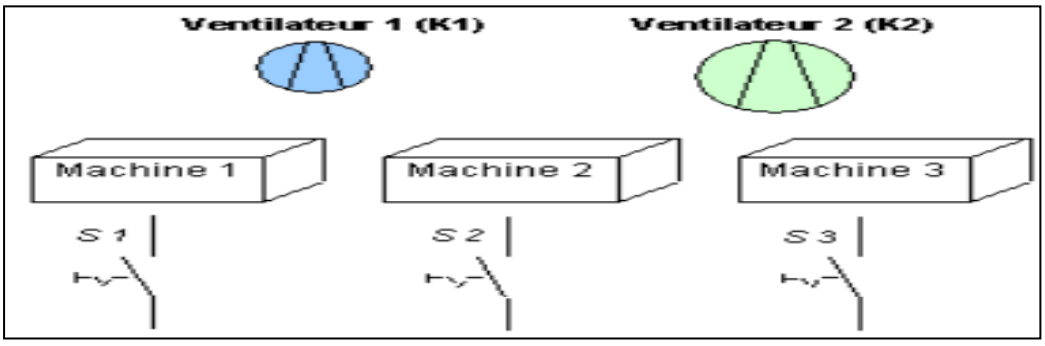
a. Dès qu'un ou plusieurs interrupteurs sont activés la lampe R doit s'allumer.
b. La lampe S ne doit s'allumer que si au moins deux interrupteurs sont activés.

Exemple 2 : Système d'Aspiration

Lors de la fabrication des fenêtres en bois, la sciure est automatiquement aspirée par un dispositif d'aspirateur central, dont la puissance d'aspiration est fixée par les machines connectées. La puissance d'aspiration sera déterminée par l'arrêt ou la marche de deux ventilateurs (Ventilateurs 1 et 2).

Par rapport à la puissance totale P, la puissance aspirante des deux ventilateurs se répartit comme suit : Ventilateur 1: 1/3P; Ventilateur 2: 2/3P.

Le système d'aspiration est fonctionné comme suit: Si une machine est en marche, le ventilateur 1 est en marche, si les 2 machines sont en marche, le ventilateur 2 est en marche. Si les 3 machines sont en marche, les deux ventilateurs sont en marche. La mise en marche d'un ventilateur s'effectue par les contacteurs.



Dans les deux exemples précédente on vous demande de :

- Donner l'expression logique des fonctions de sorties.
- Simplifier les expressions de ces fonctions.
- Simuler les circuits logiques de fonctions simplifiées on utilisant l'EWB.

TP N°4 Introduction à l'environnement de programmation en langage C sous linux

Partie I : Découvrir Linux

- ***Objectif :***

Ce TP a pour objectifs de vous familiariser avec les opérations courantes de Linux: explorer le contenu du disque dur, édité des fichiers textes, etc ... La distribution choisie dans notre laboratoire d'informatique est la distribution de la société Red Hat: fedora 12. Il existe d'autres distributions de Linux, les plus connues sont: Ubuntu, Opensuse, Mandriva...etc

- ***Environnement graphique :***

Pour ouvrir une session, comme sur d'autres systèmes, rentrez le login et le mot de passe. Pour fermer la session, utilisez le menu en bas à gauche, option "Close la session" ("logout"). Ce menu en bas a gauche sert en outre à lancer les programmes installés. Ceux-ci se trouvent en général dans les sous-menus (Applications, Utilitaires...).

- ***Opérations courantes sur les fichiers :***

- Créez un nouveau répertoire avec votre propre nom dans votre espace Allez dans /home/LABO01/, puis choisissez le menu "Fichier/Nouveau/Répertoire..."
- Copier, Déplacer, Supprimer Ces fonctions sont présentes dans le menu contextuel. Faites plusieurs essais.
- Permissions d'accès Choisissez "Propriétés" dans le menu contextuel. Cliquez sur l'onglet "Permissions". Cette fenêtre affiche alors les permissions de lecture, d'écriture, et d'exécution du fichier, pour trois catégories d'utilisateurs : le propriétaire du fichier, dont le nom est affiché en bas, les utilisateurs faisant partie du groupe affiché, et tous les autres.

- ***Commandes de Bases – BASH –***

Un "terminal" est un "outil" permettant d'envoyer des commandes à l'ordinateur. On peut utiliser Linux dans un environnement graphique, ou bien à partir d'un terminal. Vous pouvez ouvrir un terminal dans votre environnement graphique, ou bien utiliser tout votre écran en tant que terminal. Pour ouvrir un terminal dans un environnement graphique : Ouvrez le menu Applications. (à droite en haut ou bien en bas de votre écran).

1. Dans le sous-menu Outils système, cliquez sur Terminal. Dans certaines versions, vous le trouverez dans le sous-menu Accessoires au lieu du sous-menu Outils système.

2. Pour utiliser tout votre écran en tant que terminal, appuyez sur Ctrl-Alt-F1. Pour rebasculer vers votre environnement graphique, appuyez sur Alt-F7.

3. Entrée les commandes de bases suivantes :

- ***cd*** (change directory) se déplacer dans les répertoires (essayez man cd, cd ..)
- ***ls*** (list files) afficher le contenu du répertoire courant (essayez ls -a, ls -l)
- ***pwd*** (print working directory) afficher le répertoire courant (parfois celui-ci est déjà affiché dans le prompt)

- ***mkdir*** (make directory) Créer un répertoire ou dossier.

Partie II : Introduction à la Programmation C

- ***Premier programme en C:***

Le but de ce programme est d'afficher un nombre entier
1- Ouvrer votre éditeur de texte en utilisant la commande : **kwrite TP1.c**
(Remarquer que l'extension du fichier est .petit « c » en non pas grand « C »)
(Remarque aussi que le nom de fichier « c » ne contient pas d'espace)
(Vous pouvez utiliser la commande **gedit TP1.c** si la commande **kwrite** ne marche pas avec)
2- Ecrire le code source suivant dans votre éditeur :

```
#include <stdio.h>
int a ;
int main ( ){
  printf("donnez la valeur de a :") ;
  scanf("%d",&a) ;
  printf(" la valeur du nombre est : %d \n",a) ;
return 0 ;}
```

3- Enregistrer le fichier : aller dans le menu **Fichier -> Enregistrer**
4- Compiler le programme en utilisant la commande (Terminal): **gcc TP1.c -o TP1.x (**le fichier TP1.x sera le fichier exécutable)
5- Exécuter votre programme en utilisant la commande : **./TP1.x**

- ***Exercice***

Ecrire un programme qui affiche la valeur en degrés Celsius correspondant aux valeurs 0,10,20,…,300 degrés Fahrenheit. On utilisera la formule de conversion : C=5/9(F-32)

TP N°5 Manipulation des Tableaux

Objectif

Le but de ce TP est de manipuler les tableaux en utilisant la boucle.

Manipulation

Partie I : notre programme doit accomplir les tâches suivantes :

1. Lire un tableau d'entiers ;
2. Trier le tableau par ordre croissant ;
3. Afficher le tableau triée;

Le programme doit respecter l'affichage suivant :

```
Entrer le nombre des éléments:5
Entrer les éléments du tableau :
2
-5
-1
20
2
Le tableau trié :
-5    -1    2    2    20
```

Partie II : Ecrire un programme qui permet de :

1. Compter les éléments non nules d'un tableau T contient N nombres.
2. Afficher le tableau
3. Trouver la valeur MIN.
4. Calculer le nombre d'occurrences (le nombre d'apparitions) d'un entier arbitrairement entré au clavier, dans le tableau T.

Une exécution de ce programme donnera à l'écran ce qui suit :

```
Donnez la dimension du tableau: 6
Remplir le tableau:
T[0]=2
T[1]=21
T[2]= -1
T[3]=3
T[4]=-9
T[5]= 0
Affichage du tableau T :
T[0]=2, T[1]=21, T[2]= -1, T[3]=3, T[4]=-9, T[5]= 0
La plus petite valeur MIN=-9
Donner un nombre : 3
Le Tableau T comporte 1 fois le nombre 3.
```

TP N°6 Fonctions et chaîne de caractères

Objectif

Ecrire un programme utilisant des fonctions et manipuler les chaînes de caractères tout en utilisant un format d'affichage imposé.

Manipulation

1. Ecrire une fonction **recherche_CH** qui renvoie la valeur (1) s'il trouve un mot dans un texte sinon elle retourne (0).
2. Le programme principal doit demander un texte (en utilisant la fonction **gets( )** pour la prise en compte les espace), et un mot et affiche le nombre d'apparition du mot dans le texte.

   Le programme doit donner l'affichage suivant :

```
Ecrire un texte :
   Tlemcen, Ecole Supérieure en Sciences Appliquées de Tlemcen 2021
Entrée un mot quelconque:
   Tlemcen
Le mot Tlemcen apparait 2 fois dans le texte.
```

Travail supplémentaire

Récrire la fonction **Recherche_CH**, pour qu'elle retourne à sa sortie la position de la première lettre du mot recherché dans le texte, et dans le cas où elle ne trouve pas le mot elle retourne la valeur -1.

Dans le programme principal on doit supprimer le mot trouver sinon il affiche le texte original.

TP N°7 Fonctions et passage par adresse

Objectif

Ecrire un programme utilisant des fonctions et la notion passage par adresse. L'idée du passage par adresse est que, pour modifier une variable par un appel de fonction, il faut passer en paramètre non pas la variable, mais un pointeur qui pointe sur la variable.

Manipulation

1. Ecrire une fonction **Calcule** qui calcule la somme et le produit des éléments d'un tableau passé en paramètre.
2. Le programme principal doit initialiser le tableau par saisie ; calcule et affiche la somme et le produit des éléments.

Le programme doit donner l'affichage suivant :

```
Donnez la dimension du tableau: 6
Remplir le tableau:
T[0]=2
T[1]=21
T[2]= -1
T[3]=3
T[4]=-9
T[5]= 0
Affichage du tableau T :
T[0]=2, T[1]=21, T[2]= -1, T[3]=3, T[4]=-9, T[5]= 0
La somme des valeurs du tableau : 16
Le produit des valeurs du tableau :0
```

TP N°8 Récursivité et programmation modulaire

Objectif

Le but de ce TP est d'écrire une fonction itérative et une fonction récursive qui permettent de calculer le PGCD de deux nombre tout en utilisant un format d'affichage imposé.

Manipulation

Ecrire un programme qui fait appel à une première fonction PGCD_iteratif qui possède comme arguments deux nombres entiers a et b, cette fonction doit permettre de calculer le PGCD de a et b de façon itératif en utilisant l'algorithme d'Euclide qui consiste en ce qui suit :

```
Début
 Rèpèter
r = a mod b;
a = b;
b = r;
Tant que (r ≠ 0);
    PGCD(a,b) = a;
Fin
```

Le même programme devra faire appel à une deuxième fonction PGCD_recursif qui permet de calculer le PGCD de deux nombre reçus en arguments par récursivité.

Exemple :

```
Entrez la valeur de a = 77638
Entrez la valeur de b = 9842
pgcd en itératif = 2
pgcd en récursif = 2
```

II. Réécrire l'une des fonctions précédentes dans un fichier entête nommé (pgcd.h), ce fichier doit être inclut dans un programme source qui fait appel à cette fonction en utilisant le même format d'affichage.

TP N°9 Allocation dynamique de la mémoire (1)

Objectif

Le but de ce TP est d'écrire un programme utilisant l'allocation dynamique de la mémoire tout en utilisant un format d'affichage imposé.

Manipulation

Ecrire un programme qui demande à l'utilisateur de rentrer une chaîne de caractère et un caractère. Dans cette chaîne de caractère, on va "supprimer" un caractère particulier. En réalité, on crée simplement un décalage en écrasant le caractère à supprimer par le suivant. La fonction ***deleteCharInString*** ne gère pas la mémoire. Il faut donc réajuster la taille de la mémoire pour correspondre parfaitement à la nouvelle vraie taille.

Le programme doit donner l'affichage suivant :

```
Entrez une chaine de caractère :
Ceci est une chaine de caractère quelconque. On va en supprimer tous les 'e'.
Entrez le caractère de la chaine à supprimer :
e
Vous avez entre la chaine "Ceci est une chaine de caractère quelconque. On va en
supprimer tous les 'e'."  et le caractère 'e'.

La chaine après suppression de 'e' vaut maintenant :
Cci st un chain d caractr qulconqu. On va n supprimr tous ls ''.
```

TP N°10 Allocation dynamique de la mémoire (2)

Objectif

Le but de ce TP est d'écrire un programme utilisant l'allocation dynamique de la mémoire tout en utilisant un format d'affichage imposé.

Manipulation

1. Ecrire la fonction qui alloue la mémoire d'une matrice de taille « lignes*colonnes », puis qui l'initialise à la valeur val, « int ** alloue_matrice(in t lignes,int colonnes, int val) ».
2. Ecrire la fonction « void libere_matrice(int ** matrice, int n) », qui libère la matrice.
3. Afficher cette matrice pour tester vos fonctions.

Exemple :

```
Entrez le nombre de lignes : 3
Entrez le nombre de colonnes : 4
Entrez la valeur Val : 9

Affichez la matrice :
9   9   9   9
9   9   9   9
9   9   9   9
```

TP N°11 Manipulation des listes chaînées

Objectif

Le but de ce TP d'écrire plusieurs fonctions permettant de manipuler les listes chaînées. Etant donnée une liste chainée représenté par le modèle de structure suivant :

```
typedef struct cellule{
char str[100]; //la chaîne de caractères
float sup ;// la superficie
int nbr; //le nombre d'habitants
Cellule * next; //Pointeur sur la cellule suivante
 }cellule;
```

Manipulation

Soit L une liste chaînée, des villes de Tlemcen, Dont le pointeur vers la tête est T, ayant dans chaque cellule, le nom de la ville, la superficie de la ville et le nombre d'habitants dans la ville.
Écrire des fonctions ou procédures qui permettent de:

1. Récupérer toutes les villes ayant plus (strictement) de 5.000 habitants.
2. Compter le nombre de villes
3. Ajouter une ville
4. Supprimer la première ville.
5. Supprimer la dernière ville.
6. Supprimer toutes les villes ayant plus de 1.000 habitants.

TP N°12 Manipulation des listes chaînées triées

Objectif

Le but de ce TP d'écrire plusieurs fonctions permettant de manipuler les listes chaînées triées. Etant donnée une liste L de nombres entiers en représentation chaînée, écrire les fonctions en dessous.

Manipulation

1. Ecrire une fonction qui prend en paramètre une liste chaînée d'entiers et vérifie qu'elle est triée dans l'ordre croissant.
2. Ecrire une fonction qui insère un élément dans une liste chaînée triée. La liste retournée doit être triée. Pour cela, on cherchera l'adresse de la cellule (si elle existe) juste avant l'emplacement de l'insertion.

Dans le programme principal, vous créez la liste et en suivant les instructions de votre enseignant, vous testez si la liste est triée ou non, après vous affichez les éléments de la listes après insertion.

Exemple

```
Donnez le nombre des éléments de la liste: 5
Remplir la liste:
-9
-1
0
2
3
21
Donnez une valeur : 1
La nouvelle liste :
-9
-1
0
1
2
3
21
```

TP N°13 Manipulation des Piles

Objectif

Les piles définissent une structure de données de stockage qui suit une politique LIFO (Last In, First Out) d'ajout et de retrait d'élément. Un moyen simple d'implanter la structure de données des piles est d'utiliser les listes chaînées.

Manipulation

1. En supposant que les piles ont été implantées au moyen des listes chaînées, écrire les fonctions usuelles d'ajout (empiler(e,P)) et de retrait (depiler(P)) ainsi que la fonction sommet(P) qui rend le sommet de la pile et la fonction estVide(P) qui teste si la pile est vide.
2. ***Test de bon parenthésage*** : On considère une expression arithmétique dont les éléments appartiennent à l'alphabet suivant :

   $$A = \{0; \ldots ; 9; +; -; \times; \div; (;); [;]\}$$

   Écrire un programme qui permet de vérifier la validité des parenthèses et des crochets contenus dans une expression arithmétique.

TP N°14 Manipulation des Files

Objectif

Les Files définissent une structure de données de stockage qui suit une politique FIFO (First In, First Out) d'ajout et de retrait d'élément. Un moyen simple d'implanter la structure de données des piles est d'utiliser les listes chaînées.

Manipulation

Dans une gare, un guichet est ouvert. Les clients arrivent à des dates aléatoires et rentrent dans une queue. L'intervalle entre l'arrivée de deux clients successifs est un nombre aléatoire entre O et INTERVALLA_MAX (les dates sont des entiers indiquant des secondes). Lorsque le guichetier a fini de traiter un client, il appelle le client suivant dont le traitement va avoir une durée aléatoire entre 0 et DUREE_TRAITEMENT_MAX.

1. Ecrire une fonction ***creerListeClients***, qui crée une file de clients, le nombre de clients étant saisi au clavier. Cette fonction initialise aussi la date d'arrivée et la durée d'attente de chacun des clients. On supposera que le premier client est arrivé à 8h.
2. Ecrire une fonction d'affichage qui affiche le numéro de chacun des clients, sa date d'arrivée et sa date de fin de traitement en format (h min sec).

TP N°15 Manipulation des Fichiers

Objectif

Afin de pouvoir stocker des données, ou d'exploiter des données déjà existantes, il est indispensable de pouvoir manipuler des fichiers. C'est ce que nous allons voir dans ce TP.

Manipulation

Un fichier s'identifie par son nom. A partir du nom d'un fichier, on peut ouvrir un «canal» permettant de lire ou écrire dans ce fichier. Le type de ces canaux est nommé FILE. Il s'agit d'un type «abstrait», dont la définition dépend fortement du système. On ne doit donc pas chercher à regarder à l'intérieur, mais simplement appeler dessus les fonctions standards décrites dans le cours. Ces fonctions travaillent en fait toutes avec des FILE *.

1. Ecrivez une fonction compte_c(FILE * f) qui renvoie le nombre de caractères d'un fichier.
2. Ecrivez une fonction compte_m(FILE * F) qui renvoie le nombre de mots d'un fichier. Les mots sont séparés par des espaces ou des retours à la ligne. Ecrivez une fonction compte_1 qui renvoie le nombre de lignes.

Références

1. Rémy Malgouyres, Rita Zrour, Fabien Feschet. Initiation à l'algorithmique et à la programmation en C. 2e édition.
2. Robert Sedgewick. Algorithmes en langage C. 2e cycle Ecoles d'ingénieurs.
3. Clovis L.Tondo, Scott E. Gimpel. Exercices corrigés sur le langage C. 2e édition.
4.
5. Jacques LE MAITRE. Programmer en langage C : Eléments du langage et construction d'un programme avec exercices corrigés.
6. Emmanuel LAZARD. Pratique performante du langage C : Cours, techniques et exercices corrigés.
7. Vincent Granet. Mini manuel Algorithmique et de programmation.
8. Claude Delannoy. Programmer en langage C : cours et exercices corrigés. 5e édition. 2012.
9. Jean-Michel Léry. Algorithmique en C. 2e édition.
10. Cours de langage C fonctions en C. Institut d'optique graduate school
11. http://miage.univ-nantes.fr/miage/DVD-MIAGEv2/Algo_files/DVDMIAGE_Algo_Chapitre_10_Listes.pdf
12. http://staff.univ-batna2.dz/sites/default/files/bada_mosaab/files/chapitre_4_les_piles_et_les_files.pdf
13. https://chgi.developpez.com/pointeur/