الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
المدرسة العليا في الهندسة الكهربائية و الطاقوية بوهران
Ecole Supérieure en Génie Electrique et Energétique d'Oran

Manuel de Travaux Pratiques

Informatique 1 et informatique 2

Ce manuel est destiné aux étudiants de 1 ère année
des classes préparatoires en science et technologie

Réalisé par

HENDEL MOUNIA

MAITRE DE CONFERENCE A L'ESG2E

Année Universitaire 2017/2018

Avant- propos

Conforme au programme pédagogique des classes préparatoires en science et technologie, défini par arrêté ministériel, du ministère de l'enseignement supérieur et de la recherche scientifique, ce support de travaux pratiques s'adresse aux étudiants de première année, module "Informatique 1" et "Informatique 2". Ce support de TP, permet d'avoir un aperçu sur l'architecture de l'ordinateur, circuits logiques, et la programmation en langage évolué C.

TP 1: Destiné à permettre aux étudiants de connaitre et localiser les différentes composantes de l'ordinateur, ainsi que leurs caractéristiques.

TP 2: Se familiariser avec l'outil Electronics Workbench (EWB) pour la création et la manipulation de circuits logiques.

TP 3: Ecrire et exécuter un programme en C (types de base, affectation, alternatives, boucles).

TP 4: Manipulation des tableaux à une et deux dimensions et des enregistrements.

TP 5: Comprendre et manipuler les fonctions et les différents passages de paramètres de fonction.

TP 6: Comprendre et pratiquer la programmation récursive en C.

TP 7: Maîtriser les pointeurs et l'allocation dynamique sous C.

TP 8: Maîtriser la manipulation des listes chaînées, les deux structures pile et file, et les fichiers.

Sommaire

TP 5: Fonctions:

TP1: Montage et démontage d'un PC

Objectif du TP: Ce TP est destiné pour permettre aux étudiants de connaitre et localiser les différentes composantes de l'ordinateur, ainsi que leurs caractéristiques.

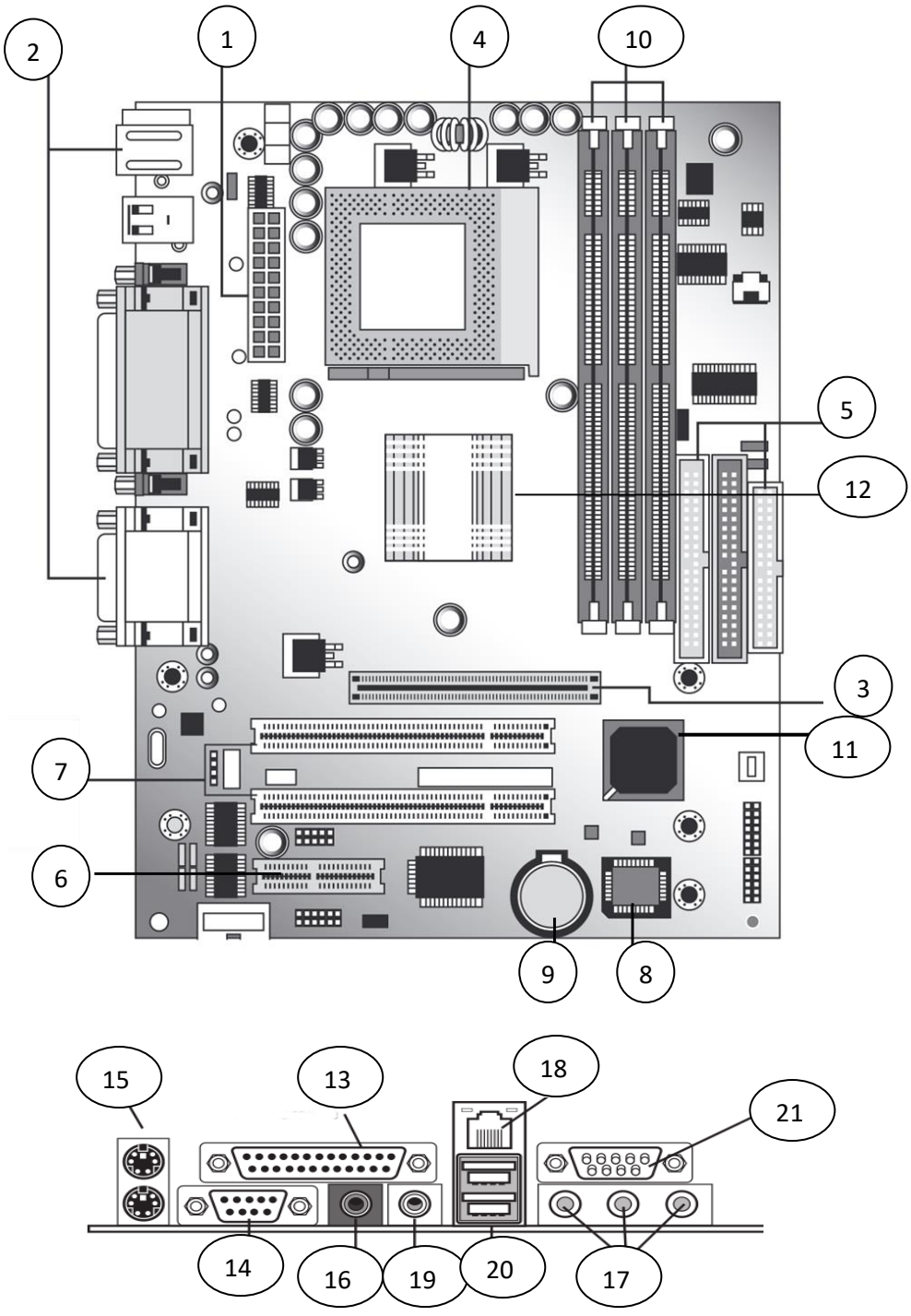


Figure 1: La carte mère.

Situer sur la figure 1 les éléments suivants de la carte mère :

1 Connecteur d'alimentation de la carte mère	11Puce South Bridge
2 Connecteurs du panneau arrière	12Puce North Bridge
3 Connecteur AGP	13Port parallèle
4 Socle processeur (socket)	14Port vidéo VGA
5 Connecteurs de disque dur et lecteur de disquettes	15Ports souris et clavier PS/2
	16Port audio numérique S/PDIF
6 Connecteurs PCI Express	17Jacks haut-parleur et microphone
7 Connecteurs PCI	18Ethernet 10/100
8 BIOS système	19Port S-Vidéo
9 Pile	20Ports USB
10 Connecteurs de mémoire	21Port série

1. La carte mère

La carte mère assure la connexion physique des différents composants (processeur, mémoire, carte d'entrées/sorties, ...) par l'intermédiaire de différents bus (adresses, données et commandes). La qualité de la carte mère est vitale puisque la performance de l'ordinateur dépend énormément d'elle. On retrouve toujours sur une carte mère :

2. Le Processeur :

Appelé également CPU (Central Processing Unit), il est le cerveau de l'ordinateur. Ce composant détermine en partie la puissance de l'ordinateur.

Il existe différents modèles et marques de processeurs, Les principales marques sont: AMD (Athlon, Sempron..) et Intel (Celeron, Pentium, I3core, I5core, I7core). On trouve également: Acer, IBM, Dell, etc.

A) Caractéristiques d'un CPU :

- La fréquence : Exprimée en gigahertz (GHz) à présent, la fréquence du processeur désigne le nombre d'opérations effectuées en une seconde par le processeur. Une horloge lui définit sa cadence.

- Le nombre de cœurs : C'est devenu le nouvel argument mis en avant par les constructeurs. Un processeur dit double cœur dispose de deux "cerveaux" indépendants, capables de traiter chacun une tâche différente. Le nombre de cœurs monte aujourd'hui jusqu'à 6 aussi bien chez AMD que chez Intel dans les processeurs grand public.

- L'HyperThreading : Il s'agit ici de pouvoir faire effectuer deux tâches en même temps à un seul cœur. L'idée est de maximiser l'utilisation des cœurs en tirant le maximum de performances possibles.

- La mémoire cachee : permet de stocker une certaine quantité de données très près du processeur, évitant de recourir à la mémoire classique (plus lente).

Remarque : Le socket : c'est le nom du connecteur destiné au microprocesseur. Il détermine le type de microprocesseur que l'on peut connecter.

3. La RAM :

(Random Access Memory) ou mémoire vive, elle contient tous les programmes en cours d'exécution ainsi que leurs données. La RAM est une mémoire volatile ce qui signifie qu'elle perd son contenu dés qu'elle n'est plus alimentée.

A) Caractéristiques de la RAM :

- **Le temps d'accès** : correspond au temps entre la demande de lecture/écriture et la disponibilité de la donnée.
- **Le temps de cycle** : correspond à l'intervalle minimum qu'il faudra entre deux accès successifs.
- **Débit** : le volume d'informations échangées par unité de temps.
- **Capacité** : plus un ordinateur a de mémoire plus il effectue les calculs rapidement.

B) Types de RAM :

- **DRAM EDO** (Extended Data Out) : utilisée dans les ordinateurs au début des années 1990. N'étant pas capable de supporter des fréquences supérieures à 66 MHz, ce type de mémoire a disparu au profit de la mémoire SDRAM.
- **SDRAM (**synchronousDRAM): ce type de mémoire existe dans différentes fréquences (66, 100, 133 et 150 MHz). Nous les utilisons avec les processeurs Pentium 1, 2 et 3, les Intel Celeron, les processeurs AMD K6 et Duron. Les SDRAM 133 MHz sont disponibles en 128, 256 et 512 Mb.
- **DDR-SDRAM (**Double Data Rate) : est une variante de la SDRAM dans laquelle on effectue deux transferts par cycle d'horloge. (fréquences de 100 MHz à 300 MHz). Nous les utilisons avec les processeurs Intel Pentium 4 et les processeurs AMD Athlon.
- **DDR2-SDRAM** : La fréquence du bus est double de celle des **DDR-SDRAM** ce qui double une fois de plus la bande passante. Ce type de mémoire a été utilisé avec les processeurs sortis entre 2004 et 2009
- **DDR3-SDRAM** : a succédé à la DDR2 en 2007 en doublant une fois de plus le taux de transfert par rapport à la génération précédente.

4. Le chipset :

C'est une interface d'entrée/sortie. Elle est constituée par un jeu de plusieurs composants chargés de gérer la communication entre le microprocesseur et les périphériques. Il est composé par deux composants baptisé **Pont Nord** et **Pont Sud**. Le pont Nord s'occupe d'interfacer le microprocesseur avec les périphériques rapides (mémoire et carte graphique) alors que le pont sud s'occupe d'interfacer le microprocesseur avec les périphériques plus lents (disque dur, CDROM, lecteur de disquette, réseau, etc…).

5. le BIOS (Basic Input Output Service) :

C'est un programme responsable de la gestion du matériel : clavier, écran, disques durs etc... Il est sauvegardé dans une mémoire morte (EEPROM) et agit comme une interface entre le système d'exploitation et le matériel.

6. Les ports de connexion :

Ils permettent de connecter des périphériques sur les différents bus de la carte mère. Il existe des ports « internes » pour connecter des cartes d'extension (PCI Express, PCI, ISA, AGP) ou des périphériques de stockage (IDE, Serial ATA) et des ports « externes » pour connecter d'autres périphériques (série, parallèle, USB, etc …).

A) Les ports internes :

- Bus ISA (Industry Standard Architecture): C'est l'ancêtre du bus PCI. On ne le retrouve plus sur les nouvelles générations de cartes mères.

- Bus PCI (Peripheral Component Interconnect) : créé en 1991 par Intel. C'est le premier bus à avoir unifié l'interconnexion des systèmes d'entrées/sorties sur un PC et à introduire le système plug-and-play. Il autorise aussi le DMA. C'est un bus de 32 ou 64 bits. On retrouve une révision du bus PCI sur les cartes mères de serveurs ayant une largeur de bus de 64bits et une fréquence de 133 MHz.

- Bus AGP (Accelered Graphic Port) : créé en 1997 lors de l'explosion de l'utilisation des cartes 3D qui nécessitent toujours plus de bandes passantes pour obtenir des rendus très réalistes. C'est une amélioration du bus PCI. Il autorise en plus le DIME (DIrect Memory Execution) qui permet au processeur graphique de travailler directement avec les données contenues dans la RAM sans passer par le microprocesseur à l'instar d'un DMA. C'est un bus 32 bits et son débit maximum est de 2 Go/s (en X8).

- Le PCI Express, remplaçant des bus PCI et AGP, permet d'atteindre des débits de250 Mo/s dans sa version de base qui peuvent monter jusqu'à 8Go/s dans sa version X16 destinée à des périphériques nécessitant des bandes passantes très élevées (applications graphiques).

- Bus IDE : il permet de relier au maximum deux périphériques de stockage interne par canal (disque dur ou lecteur DVDROM/CDROM). Son débit est de 133 Mo/s. Lorsque deux périphériques sont reliés sur le même canal, un doit être le maître (prioritaire sur la prise du bus) et l'autre l'esclave.

- Le Serial Ata, remplaçant du bus IDE, présente des débits atteignant 300 Mo. Il permet de connecter des disques durs ou des lecteurs optiques.

B) Les ports externes :

- Bus USB (Universal Serial Bus): C'est un bus d'entrée/sortie plug-and-play série. Dans sa deuxième révision (USB 2.0), il atteint un débit de 60 Mo/s. Un de ces avantages est de pouvoir connecter théoriquement 127 périphériques. Il supporte de plus le hot plug-and-play (connexion ou déconnexion de périphériques alors que le PC fonctionne).

- Port série: Le port série supporte le standard RS-232, créé en 1960 avec un seul objectif qui est de permettre la communication entre un terminal (Data Terminal Equipment, DTE ; un ordinateur) et un appareil communiquant (Data Communications Equipment, DCE ; un modem).
- Port parallèle: Le Port parallèle est un connecteur situé à l'arrière des ordinateurs compatibles PC reposant sur la communication parallèle. La communication parallèle a été conçue pour une imprimante imprimant du texte, et des images.

7. La carte vidéo

Le rôle de la carte graphique est de convertir les données numériques à afficher en un signal compréhensible par un écran. La carte vidéo communique avec la mémoire centrale et le microprocesseur par l'intermédiaire d'un bus. Actuellement, c'est le bus PCI Express qui présente des débits de 8 Go/s. Alors qu'à ses débuts, la carte vidéo se chargeait uniquement d'afficher une simple image formée de points colorés (pixel), les derniers modèles apparus se chargent d'afficher des images en 3D d'une grande complexité. C'est donc un système à microprocesseur à elle seule qui est composée par :

- Le GPU : (graphic processor unit): il est le processeur central de la carte graphique. Il se charge du traitement des données vidéo, permettant ainsi de soulager le microprocesseur. Son rôle est de traiter les objets 2D ou 3D d'une scène envoyées par le microprocesseur puis d'en déduire les pixels de l'image à afficher (projection de la scène 3D dans une grille de pixels).

- La mémoire vidéo : Elle sert à stocker les images et les textures à afficher. Elle doit présenter des débits très importants. Actuellement, la plupart des cartes graphiques sont dotées de DDR SDRAM.

- Le RAMDAC : (Random Access Memory Digital AnalogConverter) convertit les signaux délivrés par la carte en signaux analogiques compatibles avec la norme VGA des moniteurs.

TP 2: Initiation à la simulation de circuits logiques sous Electronics Workbench.

Objectif du TP: Se familiariser avec l'outil Electronics Workbench (EWB) pour la création et la manipulation de circuits logiques :

- Dessiner un circuit logique sous EWB.
- Convertir un circuit logique en une table de vérité ou une expression logique.
- Convertir une expression logique en un circuit logique.
- Simuler le fonctionnement d'un circuit logique en utilisant : un générateur de mots en entréé et des leds en sortie ; ou des interrupteurs en entrées et des leds en sortie.

1. Introduction : l'outil EWB

L'outil EWB est un logiciel de simulation de circuits analogiques et logiques sans avoir recours au matériel. Dans le cadre de notre TP, nous nous intéressons en particulier à la simulation et la manipulation des circuits logiques.

2. Création d'un circuit logique

Démarrer "EWB" en double cliquant sur son **icône** sur le bureau ou en cliquant sur **Démarrer**, puis **Programmes**, puis **Electronics Workbench**. Une fenêtre similaire à celle de la Figure 1 devrait apparaître.

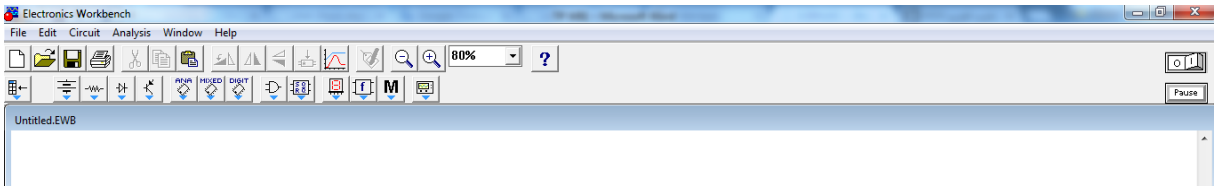


Figure 1 : barres d'outils EWB.

Pour créer des circuits logiques sous EWB, nous avons besoin d'utiliser l'onglet **''Logic gates ''**. En cliquant sur cet onglet, nous constatons deux rangées (Figure 2) :

- la première rangée contient sept portes logiques : ET, OU, NON, NOR, NAND, XOR, et XNOR.
- la deuxième rangée contient huit portes logiques commandées différemment que celles de la première rangée. (RQ : il faux ignorer la deuxième rangée).

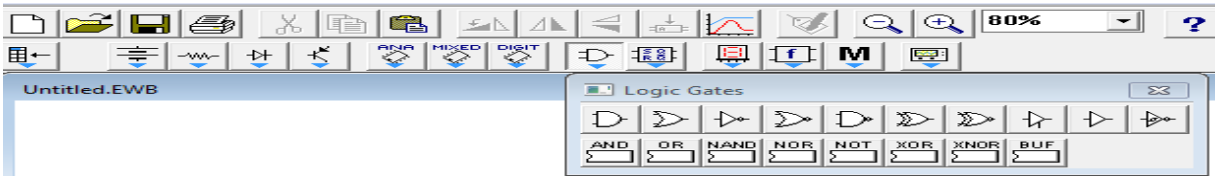


Figure 2 : Les portes logiques.

Le nombre des entrées des portes logiques est prédéfini à deux entrées. Des entrées supplémentaires peuvent être ajoutées en cliquant-droit sur la porte souhaitée, puis sur "Component Properties", puis "Number of Inputs". (Figure 3)

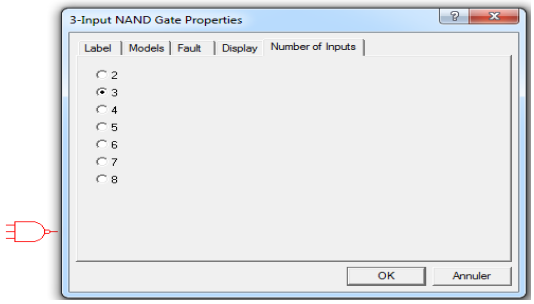
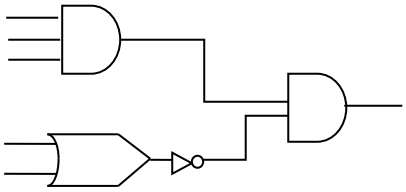


Figure 3 : Les Propriétés des composants.

- Exemple: Créer le circuit logique suivant



3. Convertisseur Logique MultiSIM :

Le convertisseur logique est utilisé pour produire la table de vérité ou l'expression booléenne d'un circuit logique auquel il est connecté. Le convertisseur logique peut également convertir une expression booléenne en une table de vérité ou un circuit logique. L'icône du convertisseur logique est représentée sur la figure 4. Notez qu'il existe huit terminaux pour le raccordement des entrées et un terminal pour la connexion de la sortie. En double-cliquant sur l'icône(a), le convertisseur logique présenté par l'icone(b) apparaît sur l'écran.

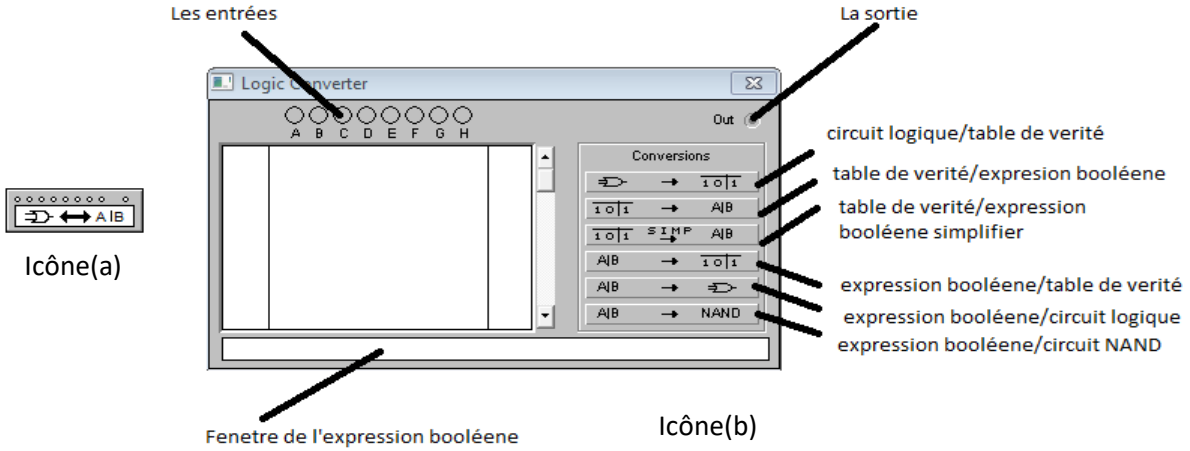
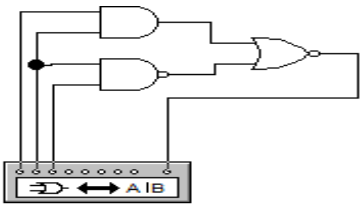
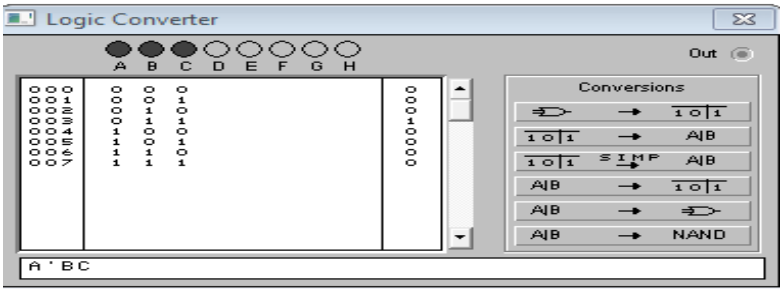


Figure 4 : Convertisseur Logique MultiSIM.

3.1 Connexion d'un circuit logique à un convertisseur logique :

- **Exemple :** Créer le circuit logique suivant, et connecter ses entrées et sa sortie au convertisseur logique.



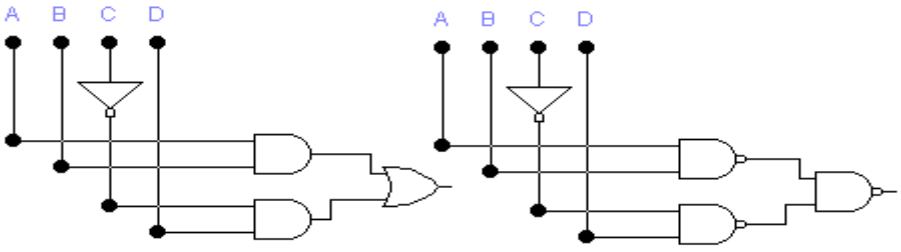


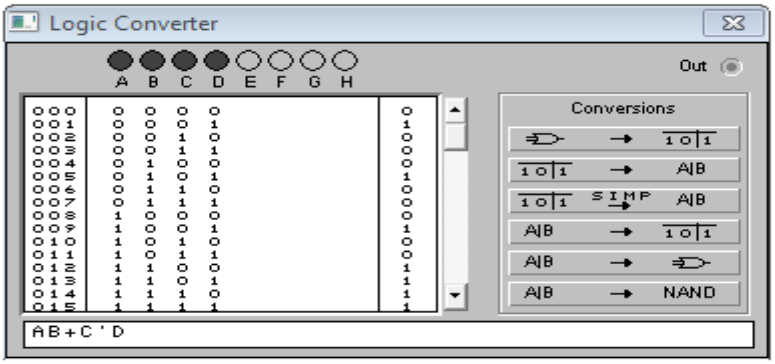
- Cliquer sur le bouton (circuit logique →table de vérité) pour voir apparaitre la table de vérité du circuit logique.
- Cliquer sur le bouton (table de vérité →expression booléen) pour voir apparaitre l'expression booléenne du circuit logique.

3.2 Création d'un circuit logique en utilisant le convertisseur logique:

En utilisant les deux derniers boutons du convertisseur logique, on peut produire un circuit logique à partir d'une expression booléenne.

- **Exemple:** Créer le circuit logique de l'expression booléen suivante: $\boldsymbol{AB + \overline{C}D}$, et le circuit logique de la même expression en utilisant les portes logique NAND.



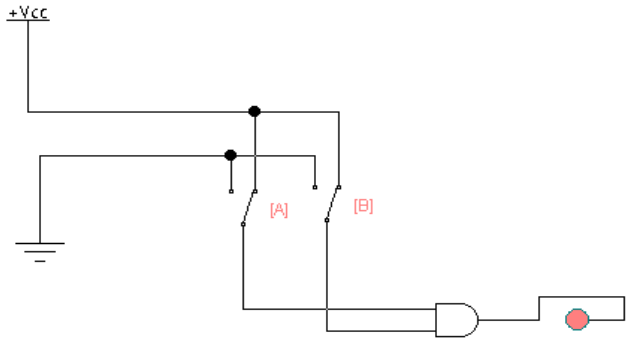


4. Simulation de la sortie d'un circuit logique

Il existe aussi deux autres méthodes (en plus de la table de vérité) pour vérifier la sortie d'un circuit logique :

4.1 Méthode 1 : Interrupteurs en entrée et Led en sortie

- **Exemple:** Réaliser le câblage suivant :

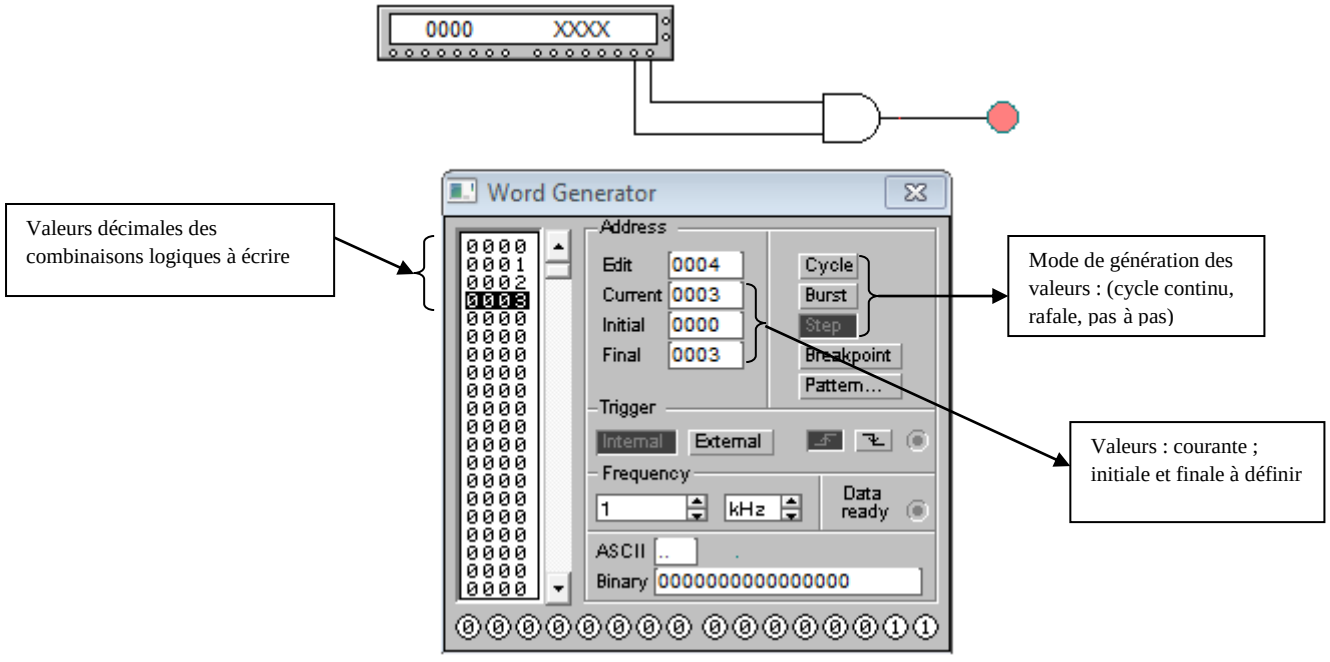


Soit une variable logique A : A(Short) ⇒A=1 et A(Open) ⇒A=0.

Pour que la led s'allume il faux mettre les deux interrupteurs A et B à **''Short ''** qui est équivalent a (A= 1 et B=1), ce qui permet de donner une valeur **''Vrai ''** en sortie de la porte logique **''ET''.**

4.2 Méthode 2 : Générateur de mots en entrée et Led en sortie.

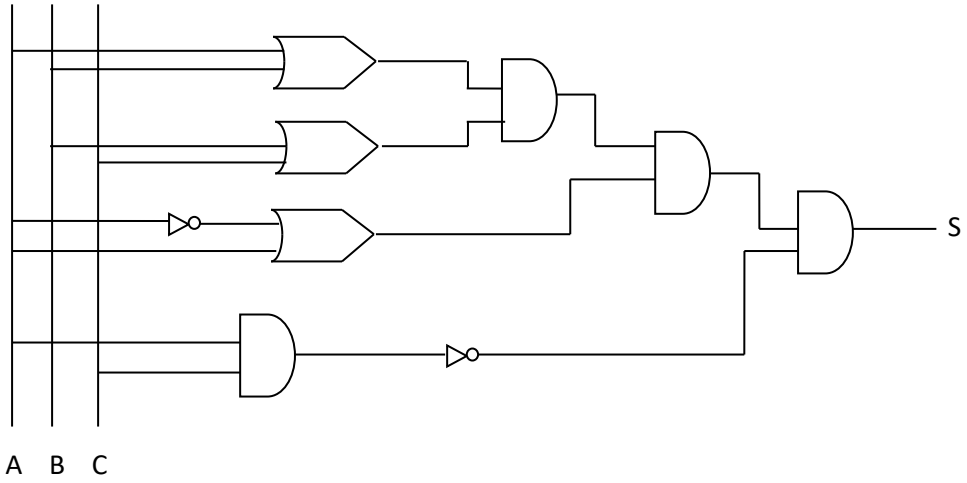
- **Exemple:** Réaliser le câblage suivant



On constate que la led s'allume quand le champ ''**Current** ''= $(3)_{10}=(11)_2$⇒(A=1 et B=1).

Exercice1 :

Réaliser sous Workbench le circuit logique suivant :

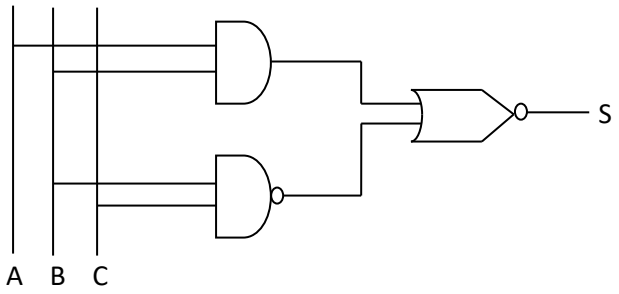


a. Ecrire la table de vérité de ce circuit. En déduire une expression équivalente pour S .
b. Donner l'expression simplifiée et le circuit correspondant en n'utilisant :
   - Que les opérateurs NON, ET et OU.
   - Que les opérateurs NON et NAND.

Exercice2 :

Vérifier la sortie du circuit suivant en utilisant les deux méthodes:

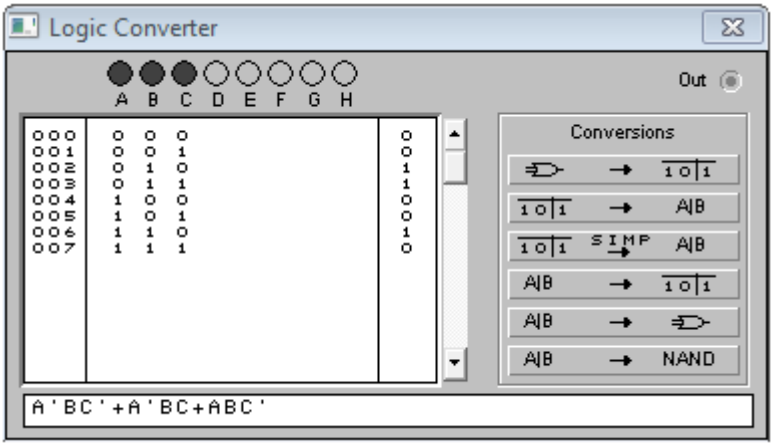
- interrupteurs en entrée et Leds en sortie.
- générateur de mots en entrée et Leds en sortie.



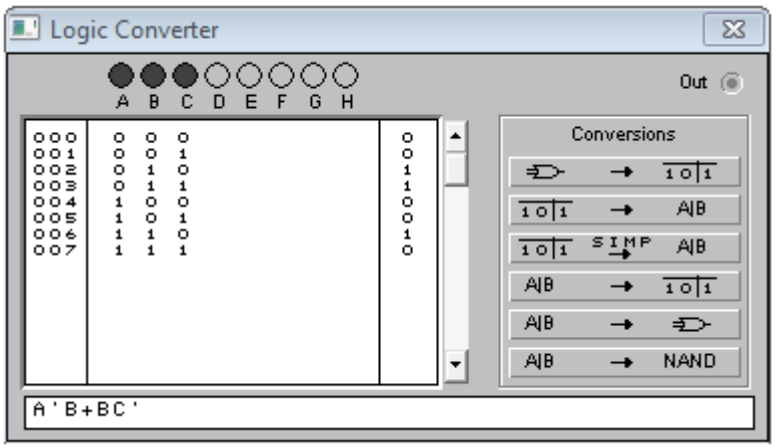
Solution du TP 2.

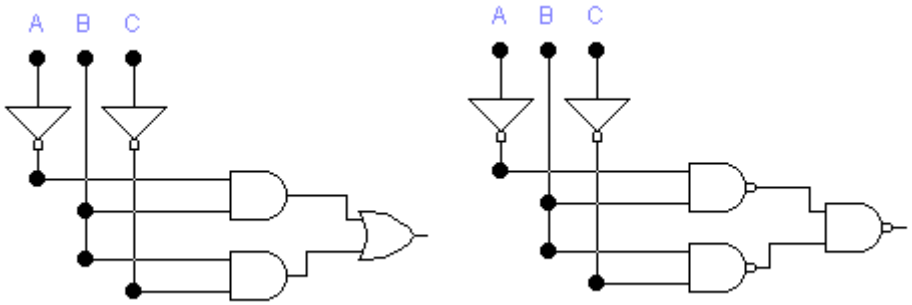
Exercice 1 :

a. Table de vérité et expression logique



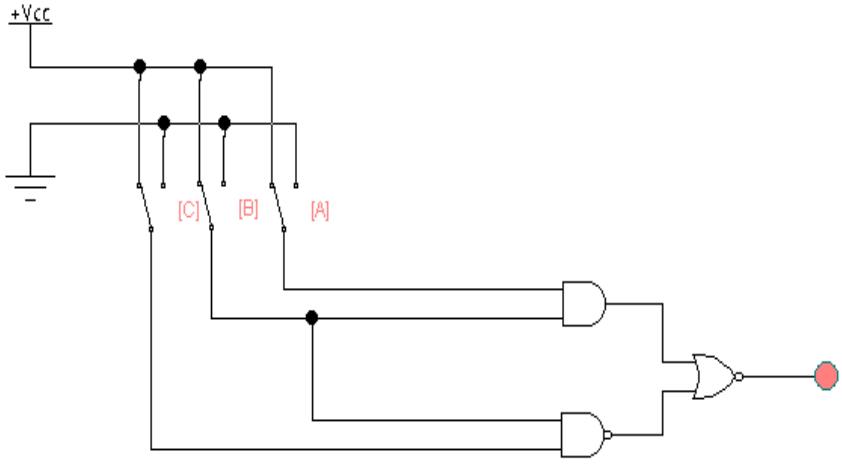
b. L'expression simplifiée et les circuits correspondant :



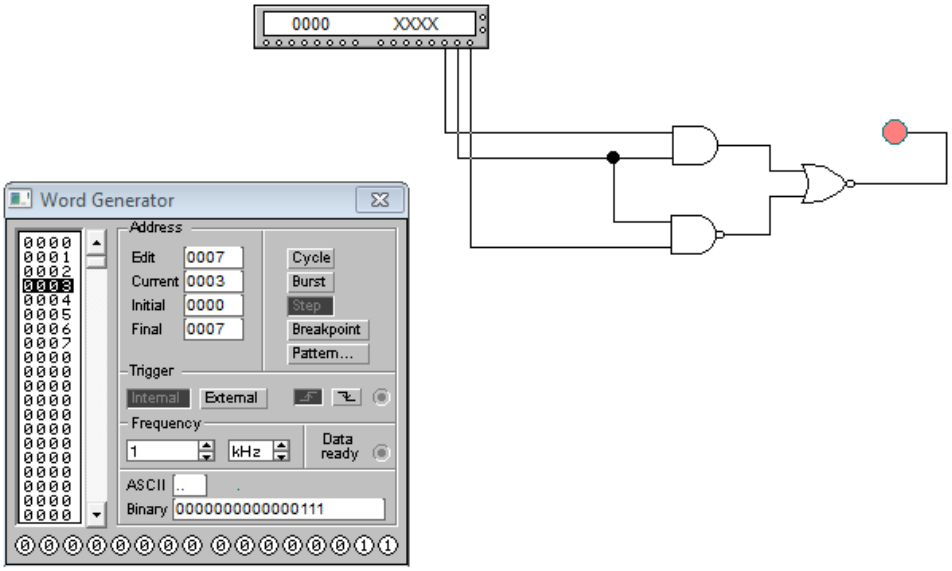


Exercice 2 :

Methode1:



Methode2 :



TP 3: Initiation au langage C.

Objectif du TP : Ecrire et exécuter un programme en C (types de base, affectations, alternatives, boucles).

La programmation a pour objectif l'automatisation de tâches à l'aide d'un programme (succession de données et d'instructions) compris et exécuté par l'ordinateur.
Au début, les programmes étaient présentés au processeur dans son langage machine (en bits ou en octets). Par la suite, on a inventé les langages assembleurs plus explicites pour les humains, et plus simples à traduire en langage compréhensif par l'ordinateur. Afin de permettre aux programmeurs d'écarter les difficultés de l'assembleur, des langages de programmation plus puissants et de plus haut niveau ont été inventés ces dernières années. Les séquences d'instructions de haut niveau (code source) sont transformées (compilées) en une séquence d'instructions machine (code objet ou binaire) par le compilateur.

Remarque :

- Les règles d'écriture des programmes et des instructions d'un langage de programmation doivent être respectées avec précision.

1. Le programme minimum

Un programme C est une suite de fonctions qui peuvent être définies dans un ordre quelconque et réparties dans plusieurs fichiers sources.

Syntaxe d'une fonction: [type] nom([typesetnomsdesparamètres]) {[corps]}

type: le type de la valeur retournée par la fonction.
corps: l'ensemble des déclarations (variables, constantes, etc.) et suite d'instructions.

Remarques :

- Tout élément placé entre crochets [] est optionnel. Le nom de la fonction, les parenthèses () et les accolades { } sont obligatoires.
- Un programme C doit obligatoirement contenir une fonction principale (main).

Compte-tenu des remarques, le programme C minimum est donc : main(){}.

2.Comment faire marcher un programme C sous Linux?

Afin d' écrire, compiler, et exécuter un programme C nommé **"minimum"**, il faut passer par les opérations suivantes:

1) Lancer la fenêtre du terminal: Ctrl+Alt+T.
2) A partir du terminal, créer le répertoire nommé **"votre_groupe "** puis se déplacer sur ce dernier:

```
$ mkdir votre_groupe        \* creer un repertoire*\
$ cd votre_groupe           \*rentrer dans votre repertoire*\
```

3) Editer un fichier contenant le programme (un programme C doit être sauvegarder sous l'extension **".c"**).

```
$ gedit minimum.c
```

4) Compiler le programme (commande gcc ou cc): deux manières de faire

$ cc minimum.c **proposition1: la compilation crée dans le répertoire courant (votre_groupe) un fichier exécutable appelé "**a.out**" **

$ cc –o mini minimum.c **proposition2: la compilation crée dans le répertoire courant (votre_groupe) un fichier exécutable appelé " **mini** " **

5) Exécuter le programme par la commande

$ ./a.out ou bien

$ ./mini (si tel est son nom).

3. Premiers programmes en C

3.1 Afficher à l'écran 'bonjour': Ce programme affiche bonjour sur l'écran du terminal plus un saut de ligne.

```
#include <stdio.h>     /* inclusion d'un fichier exterieur */
main()          /* fonction main employee sans arguments */
{
 printf("bonjour\n") ;  /* appel de la fonction printf */
}
```

Compiler et exécuter ce programme **"bonjour.c".**

Commentaires:

- Tout texte placé entre /* et */ est un commentaire ignoré au moment de la compilation.
- La ligne #include <stdio.h> est une directive au préprocesseur, qui se chargera de la remplacer par le contenu du fichier stdio.h.
- Le fichier stdio.h contient la déclaration nécessaire à l'utilisation de la fonction printf qui se charge de l' impression sur l'écran.
- {} début et fin du programme principal (main()).
- " " indiquent le début et la fin d'une chaîne de caractères.
- \n signifie passer à la ligne après avoir écrit 'bonjour'.

3.2 Additionner les nombres 3 et 5: Le programme **"addition.c"** additionne 3 et 5 et affiche le résultat.

```
#include <stdio.h>
main()
{
   int i, j, k;  /* declaration des variables i,j,k, de type entire */
   i = 3;  /* affectation de la valeur 3 à la variable i */
   j = 5 ; /* affectation de la valeur 5 à la variable j */
   k = i+j ;
   printf("Il est certain que %d+%d=%d\n",i,j,k) ;
}
```

Compiler et exécuter ce programme "**addition.c**".

Commentaires:

- Toute variable ou constante utilisée dans le programme doit être déclarée au préalable (int i: définit i comme variable de type entier signé).
- Une affectation est représentée par le signe = .
- Une fonction printf peut contenir des spécification de conversion tel %d, elle va donc remplacer le %d par le nombre entier signé qu'elle va rencontrer dans la liste des arguments qui suivent la chaîne de caractères et l'affiche en numérotation décimale. Le nombre de spécifications de conversion doit être égal au nombre des arguments(dans notre programme, aux trois spécifications de conversion %d correspondent les trois entiers i, j, k).

3.3 Saisie de données: La fonction scanf dont la déclaration est aussi contenu dans le fichier stdio.h, permet l'introduction de données (nombres, noms, ..) fournies par l'utilisateur au programme.

Syntaxe: scanf("%d",&nbr) ;

Deux points méritent l'attention :

- %d qui indique à la fonction scanf de recueillir le flux d'entrée de type entier signé.
- &nbr qui représente non pas l'identificateur nbr mais qui désigne l'adresse en mémoire de cette variable (ou le nbr doit être ecrit).

Exercice 1:

Soient les déclarations suivantes :
int n = 5, p = 9 ;
int q ;
float x ;
Quelle est la valeur affectée aux différentes variables des instructions suivantes ?

```
q = n < p ;                /* a */
q = n == p ;               /* b */
q = p % n + p > n ;        /* c */
x = p / n ;                /* d */
x = (float) p / n ;        /* e */
x = (p + 0.5) / n ;        /* f */
x = (int) (p + 0.5) / n ;  /* g */
```

Exercice 2:

Ecrivez un programme qui affiche le signe du produit de A et B sans faire la multiplication.

Exercice 3:

Écrire un programme qui :

- Lit deux entiers A et B au clavier.
- Affiche la partie entière de leur quotient.
- Affiche la partie fractionnaire frac de leur quotient.

Exercice 4:

Lors d'une opération de promotion, un magasin de composants hardware applique une réduction de 10% sur tous les composants. Écrire un programme qui lit le prix d'un composant au clavier et affiche le prix calculé en tenant compte de la réduction.

Exercice 5:

Écrire un programme qui calcule les solutions réelles d'une équation du second degré $AX^2 + BX + C = 0$, en utilisant une variable d'aide Delta pour la valeur du discriminant $B^2 - 4AC$. Puis décider si l'équation a : une $X = \frac{-B}{2A}$ ou deux $X_{1,2} = \frac{-B \mp \sqrt{D}}{2A}$ ou aucune solution réelle.
Les trois cas (A=0), (A=0 et B=0), et (A=0 et B=0 et C=0) doivent aussi être considérés en affichant les résultats et les messages nécessaires sur l'écran.

Remarque :

- Utiliser la fonction sqrt (racine carrée) de la bibliothèque <math.h> et ajouter l'option -lm à la commande de compilation.

Exercice 6:

Réécrire le programme suivant en utilisant l'instruction **switch**

```
int age ;
printf("Entrez votre age : ");
scanf("%d ", &age);
if (age = = 2)
{ printf("Salut bebe !");}
 else if (age = = 6)
 { printf("Salut gamin !");}
   else if (age = = 12)
   { printf("Salut jeune !");}
     else if (age = = 16)
     { printf("Salut ado !");}
              else if (age = = 18)
              { printf("Salut adulte !");}
                  else if (age = = 68)
                  { printf("Salut papy !");}
                      else
                     { printf("Je n'ai aucune phrase de prête pour ton âge  ");}
```

Exercice 7:

Calculez la sommation suivante:

$$S= 1 + 1/2! + 1/3! + ... + 1/N! \quad (\text{avec } N>=1).$$

a) en utilisant while.
b) en utilisant do - while.
c) en utilisant for.

Exercice 8:

1. Ecrire un programme C qui convertit un nombre binaire en son équivalent décimale.

2. Ecrire un programme C qui convertit un nombre décimal en son équivalent binaire.

Solutions du TP 3.

Exercice 1:

a) 1
b) 0
c) 5 (p%n vaut 4, tandis que p>n vaut 1).
d) 1 (p/n est d'abord évalué en int, ce qui fournit 1
e) puis le résultat est converti en float, avant d'être affecté à x).
f) 1.8 (p est converti en float, avant d'être divisé par le résultat de la conversion de n en float).
g) 1.9 (p est converti en float, avant d'être ajouté à 0.5 ; le résultat est divisé par le résultat de la conversion de n en float).

Exercice 2:

```
#include <stdio.h>
main()
{
int A, B;
printf("Introduire deux nombres (A B) non nuls : ");
scanf("%d %d", &A, &B);
if(A>0 && B<0)
printf("Le produit de A et B est negatif");
if(A<0 && B>0)
printf("Le produit de A et B est negatif");
if ((A<0 && B<0)|| (A>0 && B>0))
printf("Le produit de A et B est positif");
}
```

Exercise 3:

```
#include<stdio.h>
main()
{
int A, B;
int quotient;
float frac;
printf("Entrez deux nombres entiers non nuls: ");
scanf("%d %d", &A, &B);
quotient =A / B;
printf("Partie entière du quotient : %d\n", quotient);
frac = (A / (float)B) - quotient;
printf("Partie fractionnaire du quotient : %f\n", frac);
}
```

Exercice 4:

```
#include <stdio.h>
main()
{
```

```
float prix, reduction;
printf("Donner le prix de l'article: ");
scanf("%f", &prix);
reduction=prix*0.1;
printf("Le prix de l'article après réduction est: %f\n",prix-reduction);
}
```

Exercice 5:

```
#include <stdio.h>
#include <math.h>
main()
{
float A, B, C;
double Delta;
float X,X1,X2;
printf("Introduisez les valeurs pour A, B, et C: ");
scanf("%f %f %f", &A, &B, &C);
if (A==0)
{       if ( B==0)
                {
                 if (C==0)
                 printf("Tout réel est une solution de cette équation.\n");
                 else
                 printf("Cette équation ne possède pas de solutions.\n");
                }
        else
        {
        printf("Equation du premier degré est :\n");
        printf("  La solution X = %f\n", C/B);
        }
}
else
{
Delta= (B*B )- (4*A*C);
        if (Delta<0)
        printf("Cette équation n'a pas de solutions réelles.\n");
        else if (Delta==0)
        printf("Une seule solution réelle X = %f\n ", B/(2*A));

            else
                {
                printf("Les solutions réelles de cette équation sont :\n");
                printf(" X1 = %f\n", (B+sqrt(Delta))/(2*A));
                printf(" X2 = %f\n", (B-sqrt(Delta))/(2*A));
                }
}
}
```

Exercice 6:

```
#include <stdio.h>
main()
{
int age;
printf("Entrez votre age : ");
scanf("%d ", &age);
switch (age)
{
case 2: printf("Salut bebe !\n"); break;
case 6: printf("Salut gamin !\n");break;
case 12: printf("Salut jeune !\n");break;
case 16:printf("Salut ado !\n");break;
case 18:printf("Salut adult !\n");break;
case 68:printf("Salut papy !\n");break;
default:printf("Je n'ai aucune phrase de prête pour ton âge  \n");
}
}
```

Exercice 7:

```
/* boucle for*/

#include<stdio.h>
main()
{
float Som=0; int fact=1; int i, N;
printf("Donner N");
scanf("%d",&N);
for(i=1;i<=N;i++)
{
fact=fact*i;
Som=Som+(float)1/fact;
}
printf("Som=%f\n",Som);
}

/*boucle while */

#include<stdio.h>
main()
{
float Som=0; int fact=1; int i=1, N;
printf("Donner N");
scanf("%d",&N);
```

```
while(i<=N)
{
fact=fact*i;
Som=Som+(float)1/fact;
i=i+1;
}
printf("Som=%f\n",Som);
}
```

/* boucle do- while */

```
#include<stdio.h>
main()
{
float Som=0; int fact=1; int i=1, N;
printf("Donner N");
 scanf("%d",&N);
do
{
fact=fact*i;
Som=Som+(float)1/fact;
i=i+1;
} while(i<=N);
printf("Som=%f\n",Som);
}
```

Exercise 8:

/*convertir un nombre binaire en son équivalent décimal.*/

```
#include<stdio.h>
main()
{int nbr, p=1, nbrd=0;
printf("Donner un nombre");
scanf("%d",&nbr);
while(nbr!=0)
{nbrd=nbrd+(nbr%10)*p;
 nbr=nbr/10;
 p=p*2;
}
printf("le nombre en décimal=%d\n",nbrd);
}
```

/*convertir un nombre décimal en son équivalent binaire*/

```
#include<stdio.h>
main()
{
int nbr; int p=1; int nbrd=0;
printf("Donner un nombre");
scanf("%d",&nbr);
while(nbr!=0)
{
nbrd=nbrd+(nbr%10)*p;
 nbr=nbr/10;
 p=p*2;
}
printf("le nombre en décimal=%d\n",nbrd);
}
```

TP 4: Tableaux et Enregistrements

Objectif du TP: A l'issue de ce TP l'étudiant doit être capable de manipuler les Tableaux à une et deux dimensions et les structures enregistrements.

1. Les Tableaux:

Les tableaux représentent une collection de données homogènes (tous les éléments ont le même type) et fixes (la dimension maximale du tableau ne doit pas être dépassée), accessibles par un indice entier " i".

2. Les Enregistrements:

Les enregistrements sont des structures de données définies par le programmeur (il n'existe pas de type Enregistrement). Contrairement aux tableaux, les enregistrements peuvent contenir un nombre d'éléments finis de types éventuellement différents.

Les opérations de bases de ces structures à savoir: la déclaration, la lecture et l'écriture, seront définies au cours des exercices du TP.

Exercice 1:

Ecrire une programme qui permet d'introduire N notes dans un tableau, puis retourne: la note minimale, la note maximale et la moyenne des notes.

Exercice 2:

Ecrire un programme qui recherche une valeur donnée dans un tableau de notes de taille N. Le programme affiche la position de la valeur dans le tableau ou bien un message d'erreur si la valeur n'existe pas.

Exercice 3:

Ecrire un programme qui permet de:

- Déclarer et de remplir une matrice "Mat" de dimension [N,M].
- Calculer sa matrice transposée "Mat^T" puis l'affiche.

Exercice 4:

Nous voudrions créer un répertoire téléphonique simple, chaque enregistrement est identifié par son nom, son prénom et son numéro de téléphone.

- Déclarer un type enregistrement "personne" contenant les informations présentées à dessus.

- Écrire un programme qui permet de saisir les informations d'une personne puis les affiche.

- Déclarer le type enregistrement "manuel", qui est constitué d'un tableau de 30 enregistrements "personne", et d'un compteur qui retourne le nombre de personnes rangées dans un tableau.

- Écrire un programme qui permet d'ajouter trois personnes dans le manuel puis affiche les informations relatives à la dernière personne du manuel (nous supposons que le répertoire téléphonique contient au préalable, des enregistrements).

Solution du TP 4.

Exercice 1:

```
#include<stdio.h>
main()
{
 float Tab[50]; int N,i;
 printf("Donner la dimension N du tableau");
 scanf("%d", &N );
 for (i=0; i<N; i++)
 {
 printf("Tab[%d]",i);
 scanf("%f", &Tab[i]);
 }
float min=Tab[0];float max=Tab[0];float Som=0;
for (i=0; i<N; i++)
{
Som=Som+Tab[i];
if(Tab[i]<min) min=Tab[i];
if(Tab[i]>max) max=Tab[i];
}
float moy=Som/N;
printf("Min=%f\t Max=%f\t Moy=%f\t",min,max,moy);
}
```

Exercice 2:

```
#include<stdio.h>
main()
{
 float Tab[50]; int N,i;
 printf("Donner la dimension N du tableau");
 scanf("%d", &N );
 for (i=0; i<N; i++)
{
 printf("Tab[%d]",i);
 scanf("%f", &Tab[i]);
}
int trouve=0; float Val;
printf("Donner la valeur à rechercher : ");
scanf("%f", &Val );

for (i=0; i<N; i++)
if(Tab[i]==Val)
{
printf("La position de de la valeur %d\n",i);
trouve=1;
}
if(trouve==0) printf("la valeur n'existe pas dans le tableau\n");
}
```

Exercice 3:

```
#include<stdio.h>

main()
{
int Mat[50][50];int Mat1[50][50];int N,M,i,j;

printf("Donner nombre de lignes de la matrice");
scanf("%d", &N );
printf("Donner nombre de colonnes de la matrice");
scanf("%d", &M );

for (i=0; i<N; i++)
 for(j=0;j<M;j++)
 {
 printf("Mat[%d][%d]",i,j);
 scanf("%d", &Mat[i][j]);
 }

for (i=0; i<M; i++)
 for(j=0;j<N;j++)
Mat1[i][j]=Mat[j][i];

for (i=0; i<M; i++)
for(j=0;j<N;j++)
 printf("Mat1[%d][%d]=%d\n",i,j,Mat1[i][j]);
}
```

Exercice 4:

```
#include<stdio.h>
struct personne
{
char nom[10];
char prenom[10];
char tel[10];
};
typedef struct personne personne;
```

```
main()
{
personne pers;
printf("donner le nom de la personne:");
scanf("%s", pers.nom);
printf("donner le prenom de la personne:");
scanf("%s", pers.prenom);
printf("donner le telephone de la personne:");
scanf("%s", pers.tel);
printf("Le nom:%s\t Le prenom:%s\t Le Tel:%s\n",pers.nom,pers.prenom,pers.tel);
}
```

```
#include<stdio.h>
struct personne
{
char nom[10];
char prenom[10];
char tel[10];
};
typedef struct personne personne;

struct manuel
{
personne T[30];
int cmpt;
};
typedef struct manuel manuel;

main()
{
manuel M;int i;M.cmpt=0;
for(i=0;i<3;i++)
{
printf("donner le nom de la personne:");
scanf("%s", M.T[i].nom);
printf("donner le prenom de la personne:");
scanf("%s", M.T[i].prenom);
printf("donner le telephone de la personne:");
scanf("%s", M.T[i].tel);
M.cmpt++;
}
printf("Le nom:%s\t Le prenom:%s\t Le Tel:%s\n",M.T[M.cmpt-1].nom,M.T[M.cmpt-
1].prenom,M.T[M.cmpt-1].tel);
}
```

TP 5: Les fonctions.

Objectif du TP: comprendre et manipuler les fonctions, les pointeurs et les tableaux en C.

1. Les fonctions en C: Les fonctions permettent de structurer nos programmes en petits bouts appelés sous programmes, chacun de ces sous programmes effectue un travail bien précis. Les fonctions sont fondamentales en C, du moment ou un programme C doit contenir obligatoirement une fonction principale appelée "main".

1.1 Rappel de la structure d'une fonction:

[Type] Nom([typesetnomsdesparamètres]) {[Corps]}

Dans la définition d'une fonction, nous indiquons:

- Nom: nom de la fonction
- Le type, le nombre et les noms des paramètres de la fonction
- Type: le type du résultat fourni par la fonction
- Corps: les variables locales + les instructions à exécuter (±return).

Remarque: si la fonction ne retourne rien, Type=void.

1.2 Créer et appeler une fonction:

- Exemple: Construire une fonction triple qui permet de calculer le triple d'un nombre(en le multipliant par 3). Puis, appeler cette fonction dans un programme principal.

- Solution: deux manières de faire (avec/sans prototype):

```
#include<stdio.h>

int triple(int a) (6)
{
int resultat;
resultat=a*3;
return resultat;
}

main()   (1)
{
int nbr, res;     (2)
printf("donnez un nombre");     (3)
scanf("%d",&nbr); (4)
res=triple(nbr);          (5)/* appel de la fonction*\
printf("le triple de %d\t et %d\n",nbr,res);    (7)
}
```

valeur en entrée

valeur en sortie

```
#include<stdio.h>
int triple(int a)  /*prototype de la fonction triple*/

main()
{
int nbr, res;
printf("donnez un nombre");
scanf("%d",&nbr);
res=triple(nbr);
printf("le triple de %d\t et %d\n",nbr,res);
}

int triple(int a)
{
int resultat;
resultat=a*3;
return resultat;
}
```

- A l'appel de la fonction, l'exécution du programme est transférée à la première instruction de cette fonction.
- La fonction se termine, après l'instruction **return** ou après la dernière instruction de la fonction.
- Une fois la fonction terminée, le programme reprend à l'endroit de l'expression où l'appel avait été fait.

2. Les pointeurs en C: La spécificité du langage C vient de son traitement des pointeurs, ils sont totalement indispensables.

- **problème:** Ecrire une fonction qui renvoie deux valeurs?. **"impossible",** la fonction ne peut retourner qu'une seul valeur.

- **Solution: Utilisation des pointeurs.**

2.1 Définition d'un pointeur: Un pointeur est une variable qui contient l'adresse d'une autre variable de n'importe quel type.

- Exemple:

```
main()
{
int i=65;
int *ptr=NULL;
ptr=&i;
}
```

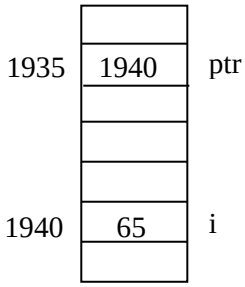


Figure 1: Représentation de l'exemple en mémoire

- **Opérateur d'adresse "&":** désigne **l'adresse** de la variable, il permet donc l'initialisation d'un pointeur.
- **Opérateur d'indirection "*":** permet d'obtenir la valeur d'un élément dont l'adresse est contenue dans le pointeur.

Exemple:

```
int i=65;
int *ptr=&i;
*ptr=*ptr+1; /*equivalent a i=i+1*/
```

- **Constante "NULL":** signifie que le pointeur ne pointe sur rien, et elle est utilisée pour des raisons de sécurité.

2.2 Envoyer un pointeur à une fonction:

- Exemple: Ecrire une fonction qui réalise la permutation entre de nombre.

a) Solution 1 erronée: sans l'utilisation des pointeur (passage par valeur)

```
#include <stdio.h>
void permutation(int a, int b)
{
int c=a;
a=b;
b=c;
}
main()
{int x,y;
printf("donnez deux nombre x, y");
scanf("%d",&x);
scanf("%d",&y);
permutation(x,y);  /*appel de la fonction*/
printf("les nouvelles valeurs de x et y sont:%d\t%d\n",x,y);
}
```

(**étape3:** permutation des copies a=15 et b=20)

(**étape1:** réserver 2 emplacements mémoire x=20 et y=15).

(**étape 2:** créer 2 copies a=20 et b=15).

(**étape 4:** affichage de x==20 et y=15).

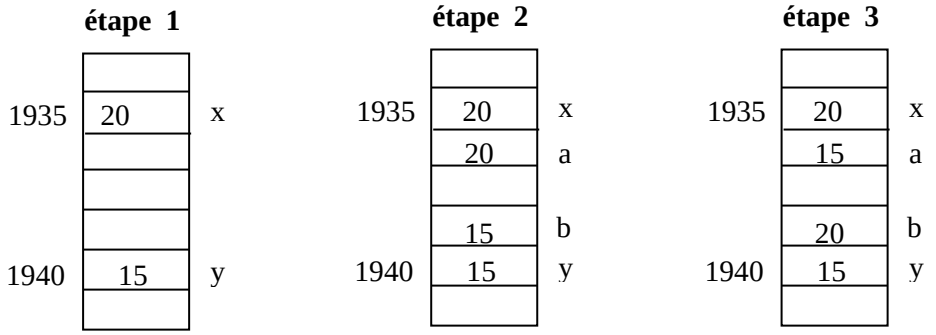


Figure 2: Représentation de la solution 1 en mémoire

b) Solution 2: utilisation des pointeur.

```
#include <stdio.h>
void permutation(int *a, int *b)
{
int c=*a;
*a=*b;
*b=c;
}
main()
{int x,y;
printf("donnez deux nombre x, y");
scanf("%d",&x);
scanf("%d",&y);
permutation(&x,&y); /*appel de la fonction*/
printf("les nouvelles valeurs de x et y sont:%d\t%d\n",x,y);
}
```

(**étape3:** permutation de *a=x=15 et *b=y=20)

(**étape1:** réserver 2 emplacements mémoire x=20 et y=15).

(**étape 2:** a= l'adresse de x et b= l'adresse de y)

(**étape 4**: affichage de x==15 et y=20).

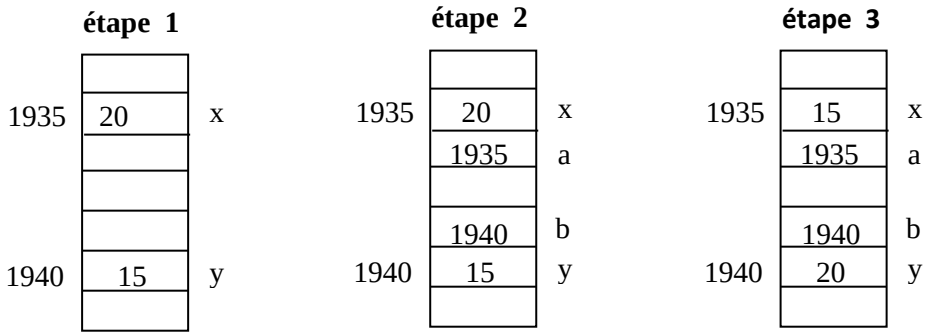


Figure 3: Représentation de la solution 2 en mémoire

- A retenir: Pour écrire une fonction qui puisse modifier une variable en utilise un pointeur. On transmet un pointeur sur la variable en question (le pointeur contiendra l'adresse mémoire de la variable), puis on change la valeur contenue à cette adresse.

2. Les pointeurs et tableaux: Le nom seul d'un tableau est un **pointeur constant** sur le premier élément de celui-ci (le pointeur contient l'adresse de la première case du tableau). Avec cette définition on peut déterminer deux manières pour manipuler les éléments d'un tableau.

- Exemple: Soit un tableau d'entiers nommé Tab contenant 10 éléments (int Tab[10]). Pour accéder a l'un de ces éléments; on peut soit utiliser l'instruction Tab[i] ou bien utiliser l'instruction *(Tab+i). Sachant que i ϵ [0,9].

Exercice1:

- Ecrire une fonction qui permet de lire la dimension N d'un tableau, ainsi que les éléments de deux tableaux A et B. (La dimension ne doit pas dépasser 15elements)
- Ecrire une fonction qui calcule le produit scalaire de deux tableaux A×B.
- Tester les deux fonctions dans un programme principal.

Exercice 2:

- Écrire, une fonction qui lit 10 nombres entiers dans un tableau puis recherche le plus grand et le plus petit élément.
- Ecrire une fonction qui fait la permutation entre deux nombres entiers.
- Ecrire une fonction qui permet de trier les éléments d'un tableau par ordre croissant (faire appel à la fonction permutation).
- Tester la première et troisième fonction dans un programme principal.

.

Solutions du TP 5.

Exercice 1:

```
#include <stdio.h>
#define Taille 15

void Lire_Tab (int *A, int*B,  int *N, int Dmax)
{
int i;
/* Saisie de la dimension du tableau */
do{
printf("Dimension du tableau (max.%d) : ", Dmax);
scanf("%d", N);
}
while (*N<=0 || *N>Dmax);
/* Saisie des composantes des 2 tableaux */
for (i=0; i<*N; i++)
{
printf("A[%d] : ", i);
scanf("%d", &A[i]);
}
for (i=0; i<*N; i++)
{
printf("B[%d] : ", i);
scanf("%d", &B[i]);
}
}

int produit_scalaire(int *A, int *B, int N)
{
int i; int Som=0;
for(i=0;i<N;i++)
Som=Som+(A[i]*B[i]);
return Som;
}
main()
{
int A[Taille], B[Taille];
int Dim;
Lire_Tab (A,B, &Dim, Taille);
printf("Le produit scalaire des deux tableau A et B=%d\n", produit_scalaire(A, B, Dim));
}
```

Exercice 2:

```
#include <stdio.h>

void fonction1(int *Tab, int *min, int *max)
{
int i;
/* Saisie des composantes du tableau */
for (i=0; i<10; i++)
{
printf("Tab[%d] : ", i);
scanf("%d", &Tab[i]);
}
*min=Tab[0];
*max=Tab[0];
for (i=1; i<10; i++)
{
if(Tab[i]<*min) *min=Tab[i];
if(Tab[i]>*max) *max=Tab[i];
}
}
void permute(int *a, int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
void Tri_ Croissant(float *Tab)
{
int i,j;
for(i=0;i<9;i++)
    for(j=i+1;j<10;j++)
    if(Tab[i]>Tab[j])   permute(&Tab[i],& Tab[j]);
}

main()
{
float Tab[10];
int i,max,min;
fonction1(Tab, &min, &max);
printf("Le Min=%d\ t et Le Max=%d\n",min,max);
Tri_ Croissant(Tab);
for (i=0; i<N; i++)
{
printf("Tab[%d]= %d\n", i,Tab[i]);
}
}
```

TP 6: La récursivité.

Objectif du TP : comprendre et pratiquer la programmation récursive en C.

1. Qu'est-ce que la récursivité ?: En informatique, une fonction est dite récursive si elle a la faculté de s'appeler elle même. L'exemple classique de la récursivité est **"la fonction factorielle":**

$$n! = 1 * 2 * 3 * \dots * (n-1) * n.$$

Sa définition récursive est la suivante:

$$\boldsymbol{f(n)} = \begin{cases} \mathbf{1} & \boldsymbol{Si\ n = 0} \\ \boldsymbol{n * f(n-1)} & \boldsymbol{Si\ n} > 0 \end{cases}$$

2. Éléments essentiels d'une méthode récursive:

- Un (ou plusieurs) cas de base: Il est obligatoire de prévoir une ou plusieurs conditions de sortie pour la fonction, sinon vous créerez une boucle infinie qui ne s'arrête jamais. Les cas de base doivent être traitès en premier lieu, si ces derniers ne sont pas vérifiés. L'appel récursif est lancé.

- Appel récursif: Il consiste en l'appel de la fonction courante. Cet appel est traité comme n'importe quel appel de fonction en C.

Exercice 1:

1) **Fonction itérative:**

- Ecrire une fonction itérative qui calcule la somme des n premiers carrés. Par exemple, si n vaut 3, cette fonction calculera: $1^2+2^2+3^2$. Cette fonction n'est définie que pour un n supérieur à 0.
- Tester la fonction dans un programme principal.

2) Passage à la fonction récursive:

- Ecrire une fonction récursive qui permet de traiter le même problème, et faire son appel dans un programme principal.

Exercice 2:

1) Ecrire une fonction récursive qui calcule le carré d'un entier, sachant que :

$$n^2=(n-1)^2+2(n-1)+1.$$

- Tester la fonction dans un programme principal.

2) Soit un tableau T:
- Ecrire une fonction récursive qui calcule la somme des éléments positifs de ce tableau.
- Ecrire une fonction récursive qui range les éléments de ce tableau en ordre inverse.
- Tester les deux fonctions dans un programme principal.

Exercice 3:

On rappelle que les nombres de Fibonacci sont définis de la façon suivante :

$$\begin{cases} F_0 = F_1 = 1 \\ F_n = F_{n-1} + F_{n-2}\ pour\ (n \geq 2) \end{cases}$$

- Ecrire une fonction récursive qui calcule le n^{ième} nombre de Fibonacci.
- Ecrire la fonction itérative équivalante.
- Tester les deux fonctions dans un programme principal.

Exercice 4:

1) Ecrire un programme qui utilise une fonction récursive permettant de calculer le coefficient binomial où n et p sont des entiers naturels. (avec $0 \leq p \leq n$.)

$$C_n^P = \begin{cases} 1 & Si\ p = 0\ ou\ p = n \\ C_{n-1}^p + C_{n-1}^{p-1} & Si\ n > p > 0. \end{cases}$$

Solution du TP 6.

Exercice 1:

1) **Fonction itérative:**

```
#include <stdio.h>
#include <math.h>
int somme_premier_carre(int n)
{
int i,S=0;
 for (i=1; i<=n; i++)
 S=S+(i*i);
return S;
}

main()
{
int n, Som;
printf("Donner un nombre positif");
scanf("%d",&n);
Som=somme_premier_carre(n);
printf("la somme des %d premiers carrés est %d\n",n,Som);
}
```

2) Passage a la fonction récursive:

```
#include <stdio.h>
#include <math.h>
int somme_premier_carre(int n)
{
if (n==1) return 1;
 else if (n>0) return (n*n)+somme_premier_carre(n-1);
}

main()
{
int n, Som;
printf("Donner un nombre positif");
scanf("%d",&n);
Som=somme_premier_carre(n);
printf("la somme des %d premiers carrés est %d\n",n,Som);
}
```

Exercice 2:

1) partie1:

```
#include <stdio.h>
#include <math.h>
int carre(int n)
{
if (n==0) return 0;
else
     if (n<0) return carre(-n);
     else return carre(n-1)+2*n-1;
}

main()
{
int nbr, c;
printf("Donner un nombre positif");
scanf("%d",&nbr);
c=carre(nbr);
printf("le carré du nombre %d est %d\n",nbr,c);
}
```

2) partie 2:

```
#include <stdio.h>
#include <math.h>
int T[50];
int somme_positif(int tab[50], int i, int N)
{
if (i>=N) return 0;
else if(i>=0)
      if(tab[i]>=0) return tab[i]+somme_positif(tab, i+1, N);
      else return somme_positif(tab, i+1, N);
}

void invers(int tab[50], int i, int N)
{
int inter;
if (i<N/2)
  {
   inter=tab[i];
   tab[i]=tab[N-i-1];
   tab[N-i-1]=inter;
   invers(tab,i+1,N);
   }
}
```

```
main()
{

int Dim,Som,j,indice=0;
printf("donner la dimention du tableau");
scanf("%d",&Dim);

for (j=0;j<Dim;j++)
{
printf("T[%d]=",j);
scanf("%d",&T[j]);
}

Som=somme_positif(T,indice,Dim);
printf("La somme des elements positifs du tableau est %d\n", Som);

invers(T,indice,Dim);
printf("Le tableau inverse est\n");

for (j=0;j<Dim;j++)
printf("T[%d]=%d\n",j,T[j]);
}
```

Exercice 3:

1) Fonction récursive:

```
#include <stdio.h>
#include <math.h>
int Fibonacci(int n)
{
 if (n==0 || n==1) return 1;
 else return Fibonacci(n-1)+ Fibonacci(n-2);
}

main()
{
int n,Fib;
printf("Donner la valeur du coefficient n de Fibonacci");
scanf("%d",&n);
Fib=Fibonacci(n);
printf("Le %d eme nombre de Fibonacci est %d\n",n,Fib);
}
```

2) Fonction itérative:

```
#include <stdio.h>
#include <math.h>
int Fibonacci(int n)
{
int tab[50];
int i;
tab[0]=1;
tab[1]=1;
for(i=2;i<=n;i++)
tab[i]=tab[i-1]+tab[i-2];
return tab[n];
}

main()
{
int n,Fib;
printf("Donner la valeur du coefficient n de Fibonacci");
scanf("%d",&n);
Fib=Fibonacci(n);
printf("Le %d eme nombre de Fibonacci est %d\n",n,Fib);
}
```

Exercise 4:

```
#include <stdio.h>
#include <math.h>
int binomial(int n, int p)
{
if (n==p || p==0) return 1;
 else return binomial(n-1,p)+ binomial(n-1,p-1);
}

main()
{
int n,p,Comb;
printf("Donner les valeurs du coefficient binomial n,p\n");
scanf("%d",&n);
scanf("%d",&p);
Comb=binomial(n,p);
printf("coefficient binomial est %d\n",Comb);
}
```

TP 7: Pointeur et Allocation Dynamique.

Objectif du TP : l'intérêt du TP consiste, d'un coté à comprendre les notions fondamentales de la programmation dynamique à savoir: l'allocation dynamique d'un espace mémoire, la vérification des erreurs d'allocation, et la desallocation de l'espace mémoire; et d'un autre coté à bien comprendre l'intérêt des pointeurs dans l'allocation dynamique.

1. Rappel sur les pointeurs: un pointeur représente une variable dont le rôle est de contenir l'adresse en mémoire d'une autre variable.

Exemple:

```
main()
{
int i=65;
int *ptr=NULL; /* déclaration d'un pointeur qui ne contient aucune adresse mémoire*/
ptr=&i;            /*ptr contient l'adresse de la variable i*/
}
```

2. Allocation dynamique:

L'allocation dynamique a pour principe de demander manuellement au système d'exploitation de nous réserver un emplacement mémoire de taille donnée. Il existe en c trois fonction principales pour la réservation de l'espace mémoire: La fonction "malloc", la fonction "calloc" et la fonction "realloc".

a) La fonction "malloc", sa syntaxe est comme suit:

ptr = malloc(nombreElements * sizeof(*ptr)).

b) La fonction "calloc", cette fonction permet la réservation mémoire, elle initialise des emplacements mémoires réservées à 0, sa syntaxe est comme suit:

ptr = calloc(nombreElements , sizeof(*ptr)).

c) La fonction "realloc", cette fonction sert à remodifier la taille d'un espace mémoire déjà réservé, sa syntaxe est comme suit:

ptr = realloc(ptr, nouvelleTaille*sizeof(*ptr)).

- La fonction "sizeof" retourne, la taille en bits de l'emplacement mémoire à allouer.
- Les fonction malloc, calloc et realloc retournent, l'adresse du premier bit de l'emplacement mémoire réservé dans le cas d'une allocation réussite, et NULL dans le cas d'un échec de réservation mémoire.

Remarque:

Il est nécessaire de tester si l'espace mémoire a bien été réservé ou pas (tester les sorties de malloc, calloc et realloc) pour éviter le problème noté "segfaults". Aussi, il est obligatoire de libérer l'espace mémoire alloué à la fin du programme en utilisant la fonction "free" dont la syntaxe est:

free(ptr).

Exemple: réservation d'un espace mémoire pour une variable entière "A " en utilisant les trois fonctions malloc, calloc et realloc.

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int *A;
A=malloc(sizeof(*A)); /* réservation de l'espace mémoire*/
if (A==NULL) exit(0); /* tester si l'espace mémoire a bien été réservé */
free(A); /*libérer l'espace mémoire réservé*/
}
```

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int *A;
A=calloc(1,sizeof(*A)); /* réservation de l'espace mémoire*/
if (A==NULL) exit(0); /* tester si l'espace mémoire a bien été réservé */
free(A); /*libérer l'espace mémoire réservé*/
}
```

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int *A;
A=malloc(sizeof(*A)); /* réservation de l'espace mémoire*/
A=realloc(A,2*sizeof(*A)); /*modifier l'espace mémoire déjà alloué*/
if (A==NULL) exit(0); /* tester si l'espace mémoire a bien été réservé */
free(A); /*libérer l'espace mémoire réservé*/
}
```

Exercice 1:

- Ecrire une fonction qui permet d'allouer dynamiquement un espace mémoire pour un tableaux de N éléments entiers et qui retourne un pointeur sur l'espace alloué.
- Ecrire une fonction qui permet d'ajouter un entier donné dans un tableau trié de dimension N.
- Tester les deux fonctions dans un programme principal.

Exercice 2:

Ecrire un programme qui permet de réaliser l'allocation dynamique d'une matrice, en appliquant trois méthodes différentes:

a - Les pointeurs sur tableaux.
b - Les tableaux de pointeurs.
c - Les doubles pointeurs.

Solution du TP 7.

Exercice 1:

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

/*Fonction pour l'allocation dynamique d'un tableau*/
void Allocation(int **Tab , int N)
{
*Tab = malloc((N+1) * sizeof(**Tab));
}
/* fonction qui permet d'ajouter un entier donné dans un tableau trie.*/
void Insert(int *Tab,int N, int Val)
{
int i;
for (i=N ; (i>0)&&(Tab[i-1]>Val) ; i--)
Tab[i]=Tab[i-1];
Tab[i]=Val;
}

main()
{
 int * Tab;int i,N,Val;
 printf("Donner la dimension N du tableau");
 scanf("%d", &N );
 Allocation(&Tab,N);
 if(Tab == NULL) exit(0);

 for (i=0; i<N; i++)
{
 printf("Tab[%d]",i);
 scanf("%d", &Tab[i]);
}

printf("Donner la valeur à insérer : ");
scanf("%d", &Val );
Insert(Tab,N,Val);

for (i=0; i<N+1; i++)
{
 printf("Tab[%d]=%d\n",i,Tab[i]);
}
}
```

Exercice 2:

1. Les pointeurs sur tableaux:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int taille1, taille2;
printf("donner le nombre de lignes");
scanf("%d",&taille1);
printf("donner le nombre de colonnes");
scanf("%d",&taille2);

int (*Mat)[taille1];     /*définir un pointeur sur un tableau de dimension "taille1"*/
/*création de "taille2" tableaux de dimension "taille1": Mat[taille1][taille2]*/
Mat=malloc(taille2*sizeof(*Mat));
if(Mat==NULL) exit(0); /* tester si l'espace mémoire a bien été réservé */
free(Mat); /*librer l'espace memoire de la matrice*/
}
```

2. Les tableaux de pointeurs.

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int taille1, taille2, i;
printf("donner le nombre de lignes");
scanf("%d",&taille1);
printf("donner le nombre de colonnes");
scanf("%d",&taille2);

int *Mat[taille1] /*définir un tableau de pointeurs de dimension "taille1"*/
/*affecter a chaque pointeur un tableau de dimension "taille2"*/
for(i=0;i<taille1;i++)
{
Mat[i]=malloc(taille2*sizeof(Mat[0]));
if(Mat[i]==NULL) /* tester si l'espace mémoire a bien été réservé*/
{
/*libérer les espaces mémoires déjà allouer*/
for(i=i-1 ; i >= 0 ; i--)
free(Mat[i]);
exit(0);
}
}
for(i=1 ; i <taille1 ; i++)  /*librer l'espace memoire de la matrice*/
free(Mat[i]);
}
```

3. Les doubles pointeurs:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int taille1, taille2, i;
printf("donner le nombre de lignes");
scanf("%d",&taille1);
printf("donner le nombre de colonnes");
scanf("%d",&taille2);

int **Mat;

/*allocation de la première dimension */
Mat=malloc(taille1*sizeof(*Mat));
if(Mat==NULL) exit(0);
/*allocation de la deuxième dimension */
for(i=0;i<taille1;i++)
{
Mat[i]=malloc(taille2*sizeof(**Mat));
if(Mat[i]==NULL) /* tester si l'espace mémoire a bien été réservé */
{
/*libérer les espaces mémoires déjà alloués*/
for(i=i-1 ; i >= 0 ; i--)
free(Mat[i]);
exit(0);
}
}
for(i=1 ; i <taille1 ; i++)  /*libérer l'espace mémoire de la matrice*/
free(Mat[i]);
}
```

TP 8: Les listes, Piles, Files, Fichier.

Objectif du TP : L'étudiant doit maitriser à la fin du TP la manipulation des listes chaînées. Afin qu'il soit capable de gérer les trois structures "pile", "file" et "fichier" .

Exercice 1: (listes chaînées)

1. Donner les déclarations nécessaires qui permettent de créer des listes chaînées de valeurs entières avec **val** comme champ de donnée et **next** comme pointeur sur le maillon suivant.

2. Écrire une fonction qui permet d'ajouter un élément à la tête d'une liste passée comme paramètre.

3. Écrire une fonction qui affiche les éléments d'une liste passée comme paramètre.

4. Écrire une fonction qui retourne la somme des valeurs d'une liste passée comme paramètre.

5. Écrire une fonction qui cherche une valeur dans une liste et retourne un pointeur sur la première occurrence de cette valeur.

6. Écrire une fonction qui supprime une valeur de la liste, la liste et la valeur sont passées comme paramètres.

7. Écrire une fonction **inserer**(..) qui insère un maillon dans une liste juste avant le premier maillon qui a une valeur supérieure à la valeur du maillon à insérer. Le pointeur sur le maillon et la liste sont passés comme paramètres.

8. Écrire une fonction qui trie une liste en utilisant un tri par insertion (utiliser la fonction **inserer**(..)).
Dans la fonction main(), présentez un menu qui affiche les traitements à réaliser sous forme d'options de choix:

```
---------------------------------------------------------------------
                    MENU PRINCIPAL
1- Ajout un élément (entier) à la liste.
2- Afficher la liste.
3- Supprimer une valeur de la liste.
4- Affiche la somme des éléments de la liste.
5- Trier la liste.
6 – Quitter.
---------------------------------------------------------------------
Taper votre choix: (1-6):
```

Exercice 2: (Piles et Files)

1. Donner les déclarations nécessaires qui permettent de créer des Piles et des Files de valeurs entières.

2. Ecrire une fonction qui permet l'empilage d'un élément dans une pile; et une fonction qui permet le dépilage d'un élément dans une pile.

3. Ecrire une fonction qui permet l'enfilage d'un élément dans une file; et une fonction qui permet le défilage d'un élément dans une file.

4. A partir des fonctions précédentes, écrire une fonction nommée "Miroir" qui détermine si un nombre donné est un *palindrome* ou pas.

5. Tester la fonction Miroir dans un programme principal.

Exercice 3: (Fichiers).

L'objectif de l'exercice est de créer un répertoire téléphonique. Une personne du répertoire est identifier par: un nom, un prénom et un numéro de téléphone.

1. Définir le type de structure personne contenant un entier et deux chaînes de caractères.
2. Définir la fonction **Ajout_Personne,** qui ajoute une personne en fin de fichier.
3. Définir La fonction **Affichage_Personne,** qui affiche l'ensemble des personnes du fichier.
4. Définir La fonction **Trouve_Tel,** qui retourne le numéro d'une personne donnée.

Exercice supplémentaire (Pile: évaluation d'une expression arithmétique post fixée)

Les expressions arithmétiques sont généralement présentées de manière infixe (l'opérateur est entre les deux opérandes). Il existe aussi la notion dite post fixée (l'opérateur est placé après ses opérandes).

Exemples: 18-20 s'écrit en post fixée 18 20 -
(20+12)*9 s'écrit en post fixée 20 12 + 9 *
-30 s'écrit en post fixée 30-

L'intérêt majeur de cette représentation est qu'il n'y a plus besoin de connaître les priorités des opérateurs, ce qui implique la non nécessité de parenthèses.
(2+7)*(5-2) s'écrit en post fixée 2 7 + 5 2 - *.

- En utilisant la notion pile écrire une fonction qui permet de retourne l'évaluation d'une expression arithmétique en notation post fixée.

Remarque: la fonction fait appel aux deux fonctions empiler et dépiler qui doivent être données au préalable.

Solution du TP 8.

Exercice 1:

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

struct liste
{
int val;
struct liste *next;
};
typedef struct liste maliste;
maliste *tete=NULL;

/* fonction qui ajoute un élément a la tête de liste */

maliste * ajouter(maliste *tete, int valeur)
{
maliste *elem = malloc(sizeof(maliste));
if (NULL == elem)
        exit(0);
elem->val = valeur;
elem->next=tete;
tete=elem;
return tete;
}

/* fonction qui affiche les éléments de la liste */

void afficher(maliste *tete)
{
maliste *pp;
printf("Affichage de la liste\n");
for(pp=tete;pp!=NULL;pp=pp->next)
printf("%d\n",pp->val);
}

/* fonction qui retourne la somme des éléments de la liste */

int Somme(maliste *tete)
{
maliste *pp; int Som=0;
for(pp=tete;pp!=NULL;pp=pp->next)
Som=Som+pp->val;
return Som;
}
```

```
/* fonction qui retourne un pointeur sur une valeur recherchée */

maliste* rechercher(maliste* tete,int valeur)
{
maliste *pp;
for(pp=tete;pp!=NULL;pp=pp->next)
if(pp-> val==valeur) return pp;
return NULL;
}

/*fonction qui supprime une valeur de la liste*/

maliste* supprimer(maliste *tete, int valeur)
{
maliste* pp,*pp1;
pp=rechercher(tete,valeur);
if (pp!=NULL)
{
        if (pp==tete)
        {
        tete=tete->next;
        free(pp);
        }
        else
        {
        pp1=tete;
                while(pp1->next!=pp)
                pp1=pp1->next;
        pp1->next=pp->next;
        free(pp);
        }
}

else printf("Elemnet introuvable\n");
return tete;
}
```

/*fonction qui insère un maillon dans une liste juste avant le premier maillon qui a une valeur supérieure à la valeur du maillon à insérer*/

```
maliste* inserer(maliste *tete, maliste *elem)
{
maliste *pp,*pp1;

/* on insert à la tete si : la liste est vide (tete==NULL) ou la valeur à insérer est inferieure ou égale à la valeur du premier maillon de la liste (tete) */

if (tete==NULL || elem->val<=tete->val)
{
elem->next=tete;
tete=elem;
}
else
{
/* on parcours la liste jusqu'à atteindre la première valeur (pp) supérieure à la valeur à insérer (pp1 est le maillon qui précède pp), si elem->val est superieure à toutes les valeurs de la liste, pp recevra NULL et pp1 pointera sur le derinier maillon de la liste */

pp=tete;
while (pp!=NULL && pp-> val<=elem->val)
        {
        pp1=pp;
        pp=pp->next;
        }
pp1->next=elem;
elem->next=pp;
}
return tete;
}
```

/*fonction qui trie les elements d'une liste (méthode de trier d'un jeu de cartes)*/

```
maliste* trier(maliste *tete)
{
maliste *pp,*tete2=NULL;
while (tete!=NULL)
{
pp=tete;
tete=tete->next; /* détache le premier maillon (pp) sans le détruire */
tete2=inserer(tete2,pp); /* l'insérer dans la nouvelle liste tete2 qui
reste toujours triée après l'ajout d'une valeur */
}
tete=tete2;
return tete;
}
```

```
/*fonction menu principal*/

void menu()
{
printf("\----------------------------------------------------------------n");
printf("\tMENU PRINCIPAL\n");
printf("1 - Ajout un élément (entier) à la liste.\n");
printf("2 - Afficher la liste.\n");
printf("3 - Supprimer une valeur de la liste.\n");
printf("4 -  Affiche la somme des éléments de la liste.\n");
printf("5 - Trier la liste.\n");
printf("6 - Quitter.\n");
printf("Taper votre choix (ex. 1): ");
}

main()
{
char choix; int valeur; int Som; maliste* pp,*pp1;
{menu();
choix=getchar();
switch(choix)
{
case '1':{printf("Donner un entier: ");scanf("%d",&valeur);
          tete=ajouter(tete,valeur); break;}
case '2':{afficher(tete);break;}
case '3':{printf("Taper la valeur a supprimer= ");scanf("%d",&valeur);
          tete=supprimer(tete,valeur);break;}
case '4':{Som=Somme(tete);
          printf("la somme=%d\n",Som);break;}
case '5':{tete=trier(tete);break;}
case '6':break;
default: printf("Choix erroné\n");
}
getchar(); /* pour lire le saut de ligne du premier getchar */
}while(choix!='6');

/*suppression de la liste*/

pp=tete;
while(pp!=NULL)
{
pp1=pp;
pp=pp>
next;
free(pp1);
}
tete=NULL;
}
```

Exercice 2:

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

struct Pile
{int val;
 struct Pile *lien;
};
typedef struct Pile mapile;
mapile *sommet=NULL;

struct File
{
int val;
struct File *lien;
};
typedef struct File mafile;
mafile * premier=NULL;

/* fonction qui empile un élément dans une pile */

mapile * empiler(mapile *sommet, int x)
{
mapile *elem = malloc(sizeof(mapile));
if(elem==NULL) exit(0);
elem->val= x;
elem->lien = sommet;
sommet = elem;
return sommet;
}

/* fonction qui depile un élément dans une pile */

mapile* depiler(mapile *sommet, int *t)
{
  if (sommet==NULL) exit(0);
  mapile * elem=sommet;
  *t=elem->val;
  sommet = sommet->lien;
  free (elem);
  return sommet;
}
```

```
/* fonction qui enfile un élément dans une file */

mafile* enfiler(mafile *premier, int x)
{
mafile *elem = malloc(sizeof(mafile));
if(elem==NULL) exit(0);
elem->val=x;
elem->lien = NULL;
if (premier==NULL)
      premier=elem;
else
    {
     mafile *elem1=premier;
     while(elem1->lien!=NULL) {elem1=elem1->lien;}
     elem1->lien=elem;
     }
return premier;
}

/* fonction qui defile un élément dans une file */

mafile* defiler(mafile *premier,int *t)
{
 if (premier==NULL) exit(0);
 mafile * elem=premier;
 *t=elem->val;
 premier = premier->lien;
 free (elem);
 return premier;
}

/*fonction Miroir*/

int Miroir(int nbr)
{
int lon=0;int N=nbr; int i,x,np,nf;
while(N!=0)
     {
      lon=lon+1;
      N=N/10;
      }
for(i=0;i<lon;i++)
     {
      x=nbr%10;
      nbr=nbr/10;
      premier=enfiler(premier,x);
      sommet=empiler(sommet,x);
     }
```

```
for(i=0;i<lon/2;i++)
{
sommet=depiler(sommet,&np);
premier=defiler(premier,&nf);
 if(np!=nf) return 0;
}
return 1;
}

/* function principal main()*/

main()
{
int nbr;
printf("Donner un nombre");
scanf("%d",&nbr);
int res=Miroir(nbr);
if(res==1)
   printf("nombre palindrome \n");
else
   printf("nombre non palindrome \n");
}
```

Exercice 3:

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

typedef struct
{
  char nom[10];
  char prenom[10];
  int tel;
} personne;

/* Fonction "Ajout_Personne" */

void Ajout_Personne (FILE *rep)
{
  personne pers;
  printf ("nom : ");
  scanf ("%s", pers.nom);
  printf ("prenom : ");
  scanf ("%s", pers.prenom);
  printf ("tel : ");
  scanf ("%d", &pers.tel);
  fwrite (&pers, sizeof(pers), 1, rep);
}
```

/* Fonction "**Affichage_Personnes**" */

```
void Affiche_Personnes(FILE *rep)
{
   personne pers;
   fseek(rep,0,SEEK_SET);
   while (fread(&pers,sizeof(personne),1,rep) != 0)
  {
   printf("nom .....: %s\n",pers.nom);
   printf("prenom ..: %s\n",pers.prenom);
   printf("tel .....: %d\n\n",pers.tel);
   }
}
```

/* Fonction **"Trouve_Tel"** */

```
void Trouve_Tel(FILE *rep, char *nom) {
 personne pers;
 int trouve = 1;
 printf("Donnez le nom : ");
 scanf("%s",nom);
 fseek(rep, 0, SEEK_SET);
 while (fread(&pers, sizeof(personne), 1, rep) != 0 && trouve != 0)

    {
     if ((trouve = strcmp(pers.nom, nom)) == 0)
     printf("Tel de %s %s : %d\n\n",pers.prenom, pers.nom, pers.tel);
     }
 if (trouve != 0)
 {
 printf("Ce nom n'existe pas\n");
 }
}
```

```
/* menu principale */

void menu(FILE *rep)
{
 char reponse, nom[10];
 do
 {
   printf("AJOUTER.... : A\n");
   printf("LISTER..... : L\n");
   printf("CHERCHER....: T\n");
   printf("QUITER......: Q\n");
   printf("  votre choix: ");
   scanf(" %c",&reponse);


   switch(reponse)
  {
     case 'a':case 'A': Ajout_Personne (rep);
     break;
     case 'l':case 'L': Affiche_Personnes(rep);
     break;
     case 't':case 'T': Trouve_Tel(rep, nom);
   }
}
while( reponse != 'q' && reponse != 'Q');
}

main()
{
FILE *repertoire;
repertoire = fopen ("ListeTel.dat", "a+");
menu(repertoire);
fclose (repertoire);
}
```

Bibliographie

[1] E. Malgouyres , R. Zrour et F.Feschet , "Initiation à l'Algorithmique et à la Programmation en C", 2eme édition DUNOD, 2011.

[2] C. Delannoy, "Exercices en langage C++ ", 3em édition EYROLLE, 2007.

[3] George Watson, Univ. of Delaware, 1999. (http://www.physics.udel.edu/~watson/scen103/lab/) .

[4] (http://e-learning.tsu.ge/course/view.php?id=1110)

[5] (http://www.choixnumerique.com/processeur)

[6] (http://eventus-networks.blogspot.com/2013/11/les-composants-de-lordinateur.html)