Séance 3

Pointeurs; Fonctions;

Pointeurs:

déclaration pointeur sur int:

```
int * p;
```

initialisation avec null:

```
p = NULL;
```

affectation avec une adresse de variable de type int

```
int a; p = &a;
```

déréférencement, opérateur *

```
int b = *p;
```

Fonctions:

déclaration:

```
int maFonction(int);
```

utilisation:

```
int x = 4;
int y = maFonction(x);
```

définition:

```
int maFonction(int a) {
     int v;
     ...
     return (v)
}
```

type de retour, arguments ou paramètres

`void`: type de retour « vide ».

variable d'entrée, variable de sortie: en C, les variables sont en entrée.

Passage de paramètre par valeur ou par adresse.

```
void maFonction(int a, int b) ;
void maFonction(int a, int * b) ;
```

Simulation d'une variable de sortie : passage de paramètres par adresse.

Une fonction peut être appelée récursivement.

Macro:

```
#define MA_MACRO(X, Y)    X + 19*Y
```

Exercices

1) Donner le schéma des cases mémoires pour chaque instruction, et corriger les.

a)
```
int A=5 ;          *p = A ;
```
b)
```
int A=1, B =5 ;    int *p1,*p2 ;      float *p3 ;
p1 = p2 ;          p2 =&A ;           *p1 = A ;
p3 = &B ;          (*p3) ++ ;
```

2) Donner la sortie du programme suivant et la structure de la mémoire en supposant qu'il effectue des
```
printf("Ligne 1: A = %d, B = %d, C = %d, *P1 = %d, *P2 = %d.\n",
A, B, C, *P1, *P2);
```
après chaque ligne.

```
int main() {
     int A=1, B=2, C=3, *P1 = &A, *P2 = &B;
     P1 = &A; P2 = &C;
     *P1 = (*P2)++;
     P1 = P2;
     P2 = &B;
     *P1 -= *P2;
     ++*P2;
     *P1 *= *P2;
     A = ++*P2**P1;
     P1 = &A;       return (0);
}
```

3) Ecrire un programme qui affiche un menu demandant à l'utilisateur de taper `A` (Addition), `S` (Soustraction), `M` (Multiplication) ou `D` (Division), qui saisit la réponse tapée au clavier (`A`, `S`, `M` ou `D`), puis qui demande deux nombres entiers tapés au clavier, et qui affiche le résultat de l'opération. On écrira une fonction pour chacun des quatre opérateurs. Enfin, on mettra ce programme dans une boucle en rajoutant la possibilité de taper `Q` pour quitter le programme.

4) Ecrire une fonction calculant factorielle N (`N!=1.2.3...(N-1).N`) et l'appeler dans un programme principal. On prendra un type `int` et on vérifiera que `N < 13` avant d'appeler la fonction. Pourquoi?

5)Programmer les fonctions `CARRE` qui reçoit en paramètre entier et lui affecte son carré, `RAZ` qui remet un entier à 0, `PLUSPLUS` qui incrémente un entier. Ecrire un programme principal qui lit un entier tapé au clavier, affiche le résultat de son passage dans `CARRE`, de son passage dans `PLUSPLUS` et de son passage dans `RAZ`.

6) Ecrire une fonction `COMPTE` comptant le nombre de fois où elle est appelée et retournant immédiatement si le nombre d'appels dépasse un certain seuil (`35` par exemple, on pourra faire un `#define SEUIL 35`). Exécuter `COMPTE 50` fois dans le `main`. Comment faire pour que, après `35` appels, le nombre d'appels reparte de `1` ?