Exercice 1

Afficher tous les caractères du jeu de caractères du langage C, avec leur code ASCII.

Corrigé

```
#include<stdio.h>

main(){
  char c;

  c=0;

  do {
    putchar(c);
    printf("   %i\n", c);
    c++;
  } while(c!=0);

}
```

Noter que les char sont signés ou non selon l'implémentation. Ils vont de -128 à 127, ou de 0 à 255. Mettre un `unsigned` en cas de désir d'éviter les char négatifs (mais pourquoi vouloir les éviter ? ?). Quoiqu'il en soit, en partant de zéro et en incrémentant, on retournera à zéro exactement quand on aura tout parcouru.

Bien noter donc la condition d'arrêt `c !=0`. D'autres choix de conditions apparemment raisonnables ne marchent pas. Par exemple, `c<=127` (ou `c<=255` en unsigned) est vrai pour tout char, et le programme bouclera. Quant à `c<127` (ou `c<255` en unsigned), ça ne bouclera pas, mais on ratera le caractère 127 (ou 255 en unsigned).

Beaucoup de caractères ne sont pas affichables. Le numéro 7 produit normalement un "bip" : moyen le plus simple de faire du bruit en langage C.

Exercice 2

1. Écrire un programme qui lit le flot de caractères tapés au clavier, et l'affiche à l'écran. Le programme doit s'interrompre quand l'utilisateur tape le caractère de fin de fichier. Ce caractère, dont le numéro est noté conventionnellement EOF est accessible à l'utilisateur depuis le clavier en tapant CTRL D (ça peut dépendre du système, sous Windows essayer CTRL Z).

2. À l'aide d'une redirection de la sortie, utiliser le programme de la question précédente pour saisir un fichier appelé texte1.

3. À l'aide d'une redirection de l'entrée, utiliser le programme de la première question pour recopier le fichier de la question précédente dans un fichier appelé copie1.

Corrigé

```
#include<stdio.h>

main(){
  char c;

  c = getchar();
  while (c!=EOF){
    putchar(c);
    c = getchar();
  }
}
```

Exercice 3

Tester le programme suivant et expliquer brièvement ce qu'il fait. Expliquer pourquoi il boucle lorsque l'utilisateur tape CTRL-D

```
#include<stdio.h>

main(){
  char c;

  do{
    c = getchar();
    putchar(c);
  } while (c!='b');
}
```

Corrigé

Le programme copie le flot d'entrée sur le flot de sortie (c'est-à-dire le clavier sur l'écran en l'absence de redirections). Le programme s'arrête quand l'utilisateur tape 'b'. Bien noter qu'en pratique, quand l'utilisateur tape 'b' rien ne se passe. Il faut qu'un événement quelconque vide le buffer (typiquement, l'utilisateur tape ENTER, mais ça peut dépendre du système).

Le programme boucle sur mon mac avec CTRL D, j'espère qu'il en est de même en salle machine ... Quand on tape CTRL D, on tape le caractère fin de fichier. Donc, le fichier "clavier", ou plus techniquement flot d'entrée standard, est FERMÉ. Donc, les getchar() suivants valent tous EOF, indépendemment de ce qu'on peut taper au clavier ... Joie des entrées/sorties.

De ce qui précède, on peut conclure que `while (c !='b')` est quasiment une faute de programmation. Si on veut s'arrêter au premier 'b' rencontré, `while (c !='b' && c !=EOF)` est préférable.

Exercice 4

Le but de cet exercice est d'écrire un programme qui lit l'entrée et établit quelques statistiques simples. Tester ce programme au clavier, puis en redirigeant un fichier vers l'entrée, par

exemple le fichier ballade.text disponible sur le site web du langage C.

1. Nombre de caractères.

2. Nombre de lettres majuscules.

3. Nombre de chiffres.

4. Nombre de lignes.

5. Longueur de la plus longue ligne.

Corrigé

On s'autorise à utiliser certaines particularités du code ASCII : les lettres majuscules sont consécutives, les lettres minuscules idem, et les chiffres aussi. Le résultat est un programme théoriquement non portable, et les puristes préferront utiliser des fonctions de la bibliothèque comme `isletter`, `isdigit`, etc.

Pas de difficulté notable, sauf la définition précise de la longueur d'une ligne. Le caractère de fin de ligne en fait-il partie ? Question de peu d'intérêt : adopter une convention et s'y tenir.

```
#include<stdio.h>

main(){
  char c;

  int nb_char, nb_maj, nb_chiffre, nb_ligne, ligne_long;
  int i; //Ou on en est dans une ligne

  nb_char = 0;
  nb_maj = 0;
  nb_chiffre = 0;
  nb_ligne = 0;
  ligne_long = 0;


  c = getchar();
  while (c!= EOF){

    nb_char++;
    i++; // On avance dans la ligne
    if ('A' <= c && c<= 'Z') nb_maj++;
    if ('0' <= c && c<= '9') nb_chiffre++;
    if (c=='\n') {
      nb_ligne++;
      if (i>ligne_long) ligne_long = i-1;
      i=0; // On repart a zero dans la nouvelle ligne
    }
```

```
    c = getchar();

  }

  //Le nombre de lignes est faux sauf en cas de derniere ligne vide
  if (i>ligne_long) ligne_long=i-1;
  if (i>0) nb_ligne++;

  //Affichage des resultats


  printf("nb_char : %i\n",  nb_char);
  printf("nb_maj : %i\n",  nb_maj);
  printf("nb_chiffre : %i\n",  nb_chiffre);
  printf("nb_ligne : %i\n",  nb_ligne);
  printf("ligne_long : %i\n",  ligne_long);
}
```

Exercice 5

Écrire un programme de code secret de votre choix. Ce programme lit un flot de caractères en entrée, et retourne un flot codé sur la sortie. Bien penser à écrire aussi le programme qui décode. Appliquer ce programme au codage et décodage de fichiers textes (par exemple ballade.text).

Piste à suivre la plus simple : décaler toute les lettres de 1, ou d'un entier k à fixer. Plus subtil : décaler la première lettre de 1, la deuxième de 5, la suivante de 1, puis de 5, etc ... La conception de codes secrets efficaces est une science à part entière : la cryptographie.

Corrigé

Pas de difficulté spéciale, sinon de bien rester dans le jeu de caractères affichables. Solution parmi d'autres pour décaler assez proprement une lettre minuscule c de 10 :

```
c = 'a' + ((c-'a') + 10)%26;
```

Notez qu'on obtient le célèbre code de l'avocat ('a' vaut 'k') ... Ca ne marche que pour des textes composés de lettres minuscules seulement. Pour coder des textes plus généraux, il y a des problèmes un peu embêtants : les retours à la lignes notamment.