L’exercice devra être fait sur le format suivant :

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

// Votre code

void test()
{
        // Code de test
}

int main(void)
{
        test();

        return 0;
}
```

La fonction *test()* est obligatoire et servira à tester les fonctionnalités de votre code

Détection/correction des erreurs

Dans cet exercice nous allons implémenter des codes de détection d'erreurs. L'idée est d'être capable de reconstruire une chaine de données après que celle-ci ait été modifiée par l'environnement (par exemple lors de la transmission sur un réseau ethernet). Pour cela nous vous proposons d'utiliser une structure représentant une chaine de données:

```
typedef struct Donnees Donnees;
struct Donnees
{
        char donnees[256];
        unsigned int taille;
};
```

Le champs *donnees* représente la chaine de données et le champs taille le nombre de données. Dans la chaine de données le caractère *'0'* représentera le bit *0* et le caractère *'1'*représentera le bit *1*. Par exemple, nous pouvons initialiser la chaine de données*10110101101* de la manière suivante:

```
Donnees donnees = {"10110101101", 11};
```

Nous vous allons utiliser la fonction suivante pour simuler la modification d'une chaine de données par l'environnement:

```
void simulerTransmission(Donnees * resultat, const Donnees * donnees, float
stabilite)
{
        const unsigned int max = 10000;
        unsigned int i = 0;
        resultat->taille = donnees->taille;
        for(i = 0; i < donnees->taille; i++)
        {
                if (rand() % max >= max * stabilite)
                        resultat->donnees[i] = donnees->donnees[i] == '0' ?
'1' : '0';
                else
                        resultat->donnees[i] = donnees->donnees[i];
        }

}
```

Cette fonction transforme aléatoirement la variable *donnees* vers la variable *resultat*. La variable *stabilite* permet de définir la degré de modification (entre 0 et 1) de *donnees* : plus la stabilité est élevée, plus *resultat* est semblable à *donnees*. Ainsi dans l'exemple suivant,*resultat1* est très proche de *donnees* alors que *resultat2* est très différent de *donnees*:

```
Donnees donnees = {"10110101101", 11}, resultat1, resultat2;
simulerTransmission(&resultat1, &donnees, 0.95);
simulerTransmission(&resultat2, &donnees, 0.1);
```

La première chose à faire est d'écrire une fonction pour afficher une chaine de données. Créez un fichier *detection.c* et écrivez la fonction *afficherDonnees()*, affichant une chaine de données (pas la taille):

```
void afficherDonnees(const Donnees * donnees)
{
        // Votre code
}
```

Le premier code de détection d'erreurs que nous allons expérimenter consiste à tripler les bits de la chaine de données. Dans le cas de la chaine de données *10110101101* nous produirions donc la chaine de données *111000111111000111000111111000111*. Pour décoder celle-ci il suffit alors de sélectionner le bit majoritaire pour chaque bloc de 3 bits.

Nous allons écrire la fonction *encoderTriple()*, permettant d'encoder une chaine de données et la fonction *decoderTriple()*, permettant de décoder une chaine de donnéees:

```
void encoderTriple(Donnees * resultat, const Donnees * donnees)
{
```

```
        // Votre code
}
void decoderTriple(Donnees * resultat, const Donnees * donnees)
{
        // Votre code
}
```

Nous allons maintenant étudier le principe du bit de parité. Ici le but n'est pas de corriger une chaine de données mais plutôt de détecter si elle a été modifiée par l'environnement. Le principe est le suivant: tous les blocs de *n* bits nous ajoutons un nouveau bit (un bit de parité). Ce bit est égal à *1* si le nombre de *1* du précédent bloc est pair, sinon il est égal à *0*. Pour savoir si une chaine de données a été modifiée il suffit alors de vérifier que chaque bit de parité est correct. Par exemple, pour la chaine de données *10100101101* et pour *n = 3*nous produirions la chaine de donnéees *101(1)001(0)011(1)01*.

Vous devez écrire la fonction *encoderParite()*, permettant d'encoder une chaine de données et la fonction *verifierParite()*, permettant de vérifier une chaine de donnéees:

```
void encoderParite(Donnees * resultat, const Donnees * donnees, unsigned
int n)
{
        // Votre code
}
int verifierParite(const Donnees * donnees, unsigned int n)
{
        // Votre code
}
```

Pour tester vos fonctions vous pourrez utiliser le programme de test suivant:

```
// En debut de fichier.
#include <time.h> // time()
#include <string.h> // strncmp()

void test()
{
        // A décommenter pour que l'appel a simulerTransmission() produise
une chaine de donnees aleatoire.
        //srand(time(NULL));

        Donnees donnees = {"110010001011", 12}, triple, parite,
tripleAltere, pariteAltere, tripleReconstruit;
        int resultatParite, resultatTriple;

        encoderTriple(&triple, &donnees);
        encoderParite(&parite, &donnees, 3);

        simulerTransmission(&tripleAltere, &triple, 0.8);
        simulerTransmission(&pariteAltere, &parite, 0.8);
```

```
        decoderTriple(&tripleReconstruit, &tripleAltere);
        resultatParite = verifierParite(&pariteAltere, 3);

        resultatTriple = strncmp(donnees.donnees,
tripleReconstruit.donnees, donnees.taille);

        printf("Chaine de données originale: ");
        afficherDonnees(&donnees);
        printf("\n\nAprès codage 'triple bit': ");
        afficherDonnees(&triple);
        printf("\nAprès codage 'bit de parité': ");
        afficherDonnees(&parite);

        printf("\n\nAltération du codage 'triple bit': ");
        afficherDonnees(&tripleAltere);
        printf("\nAltération du codage 'bit de parité': ");
        afficherDonnees(&pariteAltere);
        printf("\n\n");

        printf("Après décodage 'triple bit': ");
        afficherDonnees(&tripleReconstruit);
        if(resultatTriple == 0)
                printf(" == ");
        else
                printf(" != ");
        afficherDonnees(&donnees);


        printf("\n");
        if(resultatParite)
                printf("Le codage 'bit de parité' n'a pas détecté une
altération de la chaine de données originale.\n");
        else
                printf("Le codage 'bit de parité' a détecté une altération
de la chaine de données originale.\n");

}
```

La sortie de ce programme devra être la suivante:

```
Chaine de données originale: 110010001011

Après codage 'triple bit': 111111000000111000000000111000111111
Après codage 'bit de parité': 1101010000100111

Altération du codage 'triple bit': 111101000001111000100000011000011011
Altération du codage 'bit de parité': 1000010100100111

Après décodage 'triple bit': 110010001011 == 110010001011
Le codage 'bit de parité' a détecté une altération de la chaine de données
originale.
```

rapide et l'exponentiation basique.