Correction code TD3

Exercice 1

1)

```
void InitTab(char tab2D[DIM1][DIM2]){ //DIM1 pas indispensable
	int i,j;
	for(i=0;i<DIM1;i++)
		for(j=0;j<DIM2;j++)
			tab2D[i][j]=0;
}
```

2)

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#define DIM1 5
#define DIM2 6
void InitTab(char **tab2D, int nbL, int nbC){ //Autant mettre les dimensions...
	int i,j;
	for(i=0;i<nbL;i++)
		for(j=0;j<nbC;j++)
			tab2D[i][j]=0;
}

int main(void) {
	char **tab2D;
	int i,j;
	tab2D=malloc(sizeof(int*)*DIM1);
	assert(tab2D);
	for(i=0;i<DIM1;i++){
		tab2D[i]=malloc(sizeof(int)*DIM2);
		assert(tab2D[i]);
	}
	InitTab(tab2D);
	for(i=0;i<DIM1;i++)
		free(tab2D[i]);
	free(tab2D);
	return 0;
}
```

Exercice 2

1)

1. a, b et c sont les variables globales du programme, elles sont déclarées en dehors de toute fonction.
2. La fonction f n'a pas à proprement parler de variable locale. Voir "paramètres formels".
3. La fonction g a une variable locale c, celle qui est déclarée à l'intérieur de la fonction. Soulignons que cette variable locale masque la variable globale c : g ne peut plus manipuler que la variable locale c.
4. x et y sont les variables locales de main.

5. x est le paramètre formel de la fonction f. Notez que les paramètres formels sont en fait implantés sous la forme de variables locales à la fonction, initialisées lors de l'appel de la fonction. On pourrait donc dire que x est aussi une variable locale de f.
6. La fonction g n'a pas de paramètre formel.
7. Le paramètre effectif de f lors de l'appel dans main est 2 (la valeur de l'expression 'a' qui est utilisée comme paramètre lors de l'appel).
8. Cela dépend de quel c on parle. Le c global étant masqué par le c local, il est inchangé. Par contre, le c local (celui qui est retourné en résultat) a pris la valeur de a.
9. La fonction g inverse les valeurs des variables globales a et b et retourne l'ancienne valeur de a (qui est maintenant celle de b).
10. x=f(a)=f(2). Donc x=a*2=4.
Lors de l'exécution de g, la variable locale c prend la valeur de a donc 2, c'est cette valeur qui est retournée par la fonction, donc stockée dans y. a prend la valeur de b donc 3, b prend la valeur de la variable locale c donc 2.
La variable c qui est affichée est la variable globale, elle a donc toujours la valeur 4.
Le programme affiche 4 2 3 2 4.

2)
Fichier main.h
```
extern int a, b, c;
```

Fichier main.c
```
#include <stdio.h>
#include "fonction1.h"
#include « fonction2.h"

int a, b, c;

int main() {
	int x,y;
	a=2;
	b=3;
	c=4;
	x=f(a);
	y=g();
	printf("%d %d %d %d %d\n", x, y, a, b, c);
	return 0;
}
```

Fichier fonction1.c
```
#include « main.h"

int f(int x) {
	return a*x;
}
```

Fichier fonction1.h
```
int f(int x);
```

Fichier fonction2.c
```
#include « main.h"
```

```
int g() {
	int c;
	c=a;
	a=b;
	b=c;
	return c;
}
```

Fichier fonction2.h

```
int g();
```

1 Lecture et écriture de fichiers

Exercice 3

1)

```
ty_etu* lecture_ascii_etu(char *nomFi, int *nb_etu){

	FILE *pFi;
	int verif, i, j;
	ty_etu *tEtu;

	pFi=fopen(nomFi, "r");
	if(pFi==NULL){
		fprintf(stderr, "lecture_etu:: Ne peux ouvrir %s en lecture\n", nomFi);
		return NULL;
	}

	verif=fscanf(pFi, "%d", nb_etu);
	assert(verif==1);

	tEtu=malloc(sizeof(ty_etu)*(*nb_etu));
	assert(tEtu);

	for(i=0;i<*nb_etu;i++){
		fscanf(pFi, "%d %s %s %d", &tEtu[i].id_etu, tEtu[i].prenom, tEtu[i].nom,
		&tEtu[i].nb_ue);
		fprintf(stderr, "lecture_ascii_etu:: etu:%d nb ue %d\n", *nb_etu,
		tEtu[i].nb_ue);
		for(j=0;j<tEtu[i].nb_ue;j++)
			fscanf(pFi, "%s %d", tEtu[i].codes_ue[j], &tEtu[i].notes[j]);
	}

	fclose(pFi);
	return tEtu;
}
```

2)

```
void ecriture_binaire_etu(char *nomFi, ty_etu *tEtu, int nb_etu){
	FILE *pFi;
	pFi=fopen(nomFi, "w");
```

```
	if(pFi==NULL){
		fprintf(stderr, "lecture_etu:: Ne peux ouvrir %s\n", nomFi);
		return ;
	}
	// size_t fwrite(const void *restrict ptr, size_t size, size_t nitems, FILE *restrict
	stream);
	fwrite(&nb_etu, sizeof(int), 1, pFi);
	fwrite(tEtu, sizeof(ty_etu), nb_etu, pFi);

	fclose(pFi);
	return ;
}

3)
ty_etu* lecture_binaire_etu(char *nomFi, int *nb_etu){
	FILE *pFi;
	ty_etu *tEtu;

	pFi=fopen(nomFi, "r");
	if(pFi==NULL){
		fprintf(stderr, "lecture_etu:: Ne peux ouvrir %s en lecture\n", nomFi);
		return NULL;
	}

	//size_t fread(void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);
	fread(nb_etu, sizeof(int), 1, pFi);
	tEtu=malloc(sizeof(ty_etu)*(*nb_etu));
	assert(tEtu);
	fread(tEtu, sizeof(ty_etu), *nb_etu, pFi);

	fclose(pFi);
	return tEtu;
}
```

Exercice 4

1)

Cette fonction s'´ecrit très bien par récurrence. A chaque étape, on détermine la version filtrée de la liste privée de son premier élément et on remet en place le premier élément.

```
Elt *filtre_pair1(Elt *liste) {
	Elt *nelt = NULL;
	if (!liste) return NULL;
	if (liste->suivant) {
		liste->suivant = filtre_pair1(liste->suivant);
	}
	if (liste->donnee % 2) {
		nelt = liste->suivant;
		free(liste);
		return nelt;
	}
	return liste;
```

}
L'utilisation se fait de la façon suivante :
liste=filtre_pair1(liste);

2)
La liste doit être passée par pointeur, autrement dit, l'argument de la fonction sera de type Elt ** (pointeur sur pointeur sur un élément). La principale difficulté dans l'écriture de cette fonction est qu'il y a des pointeurs, des adresses, des structures, des pointeurs sur pointeurs, etc. avec donc des '&', des '*' et des '->' qui s'entremêlent.

```
void filtre_pair2(Elt **liste) {
        Elt *nelt = NULL;
        if (!(*liste)) return;
        if ((**liste).suivant) {
                filtre_pair2(&((**liste).suivant));
        }
        if ((**liste).donnee % 2) {
                nelt = (**liste).suivant;
                free(*liste);
                *liste = nelt;
        }
}
```

L'utilisation se fait de la façon suivante :
filtre_pair2(&liste);