Correction des exercices de la session 4

1. 1.1 Le type structuré peut se définir comme tel :

```
typedef struct employe_t {
    char nom[32];
    char prenom[32];
    float salaire;
} employe;
```

   1.2 Pour créer les différents employés, nous allons utiliser la fonction *strcpy* afin de facilement recopier les noms et prénoms des employés.

```
int main(){
    employe entreprise[3];
    strcpy(entreprise[0].prenom, "John");
    strcpy(entreprise[0].nom, "Doe");
    entreprise[0].salaire = 1900.0;
    strcpy(entreprise[0].prenom, "Alice");
    strcpy(entreprise[0].nom, "Renaud");
    entreprise[0].salaire = 2200.0;
    strcpy(entreprise[0].prenom, "Bob");
    strcpy(entreprise[0].nom, "Herbert");
    entreprise[0].salaire = 2300.0;
}
```

   1.3 Voici une fonction qui pourrait réaliser cette promotion :

```
void promotion(employe *emp, float augmentation) {
    emp->salaire += augmentation;
}
```

   Nous pouvons appeler cette fonction dans la suite de notre code *main* de cette manière :

```
    promotion(entreprise[0], 200.0);
```

2. 2.1 Voici comment les deux types structurés peuvent se définir :

```
typedef struct point_t {
    int x;
    int y;
} point;

typedef struct segment_t {
    point origine;
    point destination;
} segment;
```

   2.2 La fonction peut s'écrire de la façon suivante :

```
segement *nouveau_segment(int x1, int y1, int x2, int y2) {
    segment *seg;
    seg = malloc(sizeof(segment));
    if (!seg)
        return NULL;
    seg->origine.x = x1;
```

```
    seg->origine.y = y1;
    seg->destination.x = x2;
    seg->destination.y = y2;
    return seg;
}
```

2.3 Voici un code pour libérer une telle structure :

```
void liberer(segment *seg) {
    free(seg);
}
```

3. 3.1 Voici un code définissant le type structuré demandé :

```
typedef struct element_t {
    int valeur;
    struct element_t *suivant;
} element;
```

3.2 Voici un code correspondant à la fonction demandée :

```
element *ajout(element *premier, int nouvelle_valeur) {
    element *nouveau;
    nouveau = malloc(sizeof(element));
    if(!nouveau)
        return NULL;
    nouveau->valeur = nouvelle_valeur;
    if(!premier){
        premier = nouveau;
        return nouveau;
    }
    element *dernier = premier;
    while(dernier->suivant)
        dernier = dernier->suivant;
    dernier->suivant = nouveau;
    return nouveau;
}
```

3.3 Voici un code permettant de réaliser la procédure demandée :

```
void supprime(element *premier, int val) {
    element *liste;
    liste = premier->suivant;
    element *precedent;
    precedent = premier;
    if(premier->valeur == val) {
        if(premier == liste)
            free(premier)
        else {
            premier = liste;
            free(precedent);
        }
    }
    else {
```

```
        while(liste != premier){
            if(liste->valeur == val) {
                element *suiv;
                suiv = liste->suivant;
                free(liste);
                precedent->suivant = suiv;
                return;
            }
            precedent = liste;
            liste = liste->suivant;
        }
    }
}
```

3.4 Voici un code permettant de liberer tout l'espace alloué par une liste de cette structure (récursif) :

```
void liberer(element *elem) {
    if (elem->suivant)
        liberer(elem->suivant);
    free(elem);
}
```

Et voici sa version itérative :

```
void liberer(element *premier) {
    if(premier){
        element *actuel = premier;
        while(premier){
            premier = premier->suivant;
            free(actuel);
            actuel = premier;
        }
    }
}
```

4. 4.1 Voici un type structuré correspondant à ce qui est demandé dans l'énoncé :

```
typedef struct lieu_t {
    char* nom;
    struct lieu_t *prochain;
} lieu;
```

4.2 Voici un code correspondant à la fonction demandée :

```
lieu *creer_parcours(char *c) {
    if(!c){
        return NULL;
    }
    lieu *parcours;
    parcours = malloc(sizeof(lieu));
    if(!parcours)
        return NULL;
    parcours->nom = c;
```

```
    parcours->prochain = parcours;
    premier = parcours;
    while(*c) {
        if(*c == ' '){
            lieu *nouveau;
            nouveau = malloc(sizeof(lieu));
            if(!nouveau){
                parcours = premier->prochain;
                while(parcours != premier){
                    lieu *actuel = parcours;
                    parcours = parcours->prochain;
                    free(actuel);
                }
                free(premier);
                return NULL;
            }
            c++;
            nouveau->nom = c;
            nouveau->suivant = premier;
            parcours->suivant = nouveau;
            parcours = parcours->suivant;
        }
        c++;
    }
    return premier;
}
```

4.3 Voici un code permettant de libérer l'espace alloué par le type structuré créé :

```
void liberer(lieu *premier){
    if(premier){
        lieu *parcours = premier->suivant;
        while(parcours != premier) {
            lieu *actuel = parcours;
            parcours = parcours->suivant;
            free(actuel);
        }
        free(premier);
    }
}
```