Correction des exercices de la session 4

1. (a) Le type structuré peut s'écrire :

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

   (b) Le main peut s'écrire :

```
#include <string.h>
int main() {
    employe t[3];
    char *nom1 = "Doe";
    char *prenom1 = "John";
    char *nom2 = "Renaud";
    char *prenom2 = "Alice";
    char *nom3 = "Herbert";
    char *prenom3 = "Bob";
    strcpy(t[0].nom, nom1);
    strcpy(t[1].nom, nom2);
    strcpy(t[2].nom, nom3);
    strcpy(t[0].prenom, prenom1);
    strcpy(t[1].prenom, prenom2);
    strcpy(t[2].prenom, prenom3);
    t[0].salaire = 1900;
    t[1].salaire = 2200;
    t[2].salaire = 2300;
}
```

   (c) Voici la fonction et son appel dans le main :

```
void f(float *salaire, float augmentation) {
    *salaire += augmentation;
}

int main() {
    f(&(t[0].salaire), 200);
}
```

2. (a) Voici les types structurés :

```
typedef struct coord_t {
    int x;
    int y;
} coord;

typedef struct seg_t {
    coord p1;
    coord p2;
} seg;
```

(b) Voici la fonction :

```
seg *f(int x1, int x2, int y1, int y2) {
    seg *n;
    n = malloc(sizeof(seg));
    if (!n)
        return NULL;
    n->p1.x = x1;
    n->p1.y = y1;
    n->p2.x = x2;
    n->p2.y = y2;
    return n;
}
```

(c) Voici la fonction pour libérer l'espace alloué :

```
void libere(seg *p) {
    free(p);
}
```

3. (a) Voici le type structuré :

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

(b) Voici un moyen d'écrire la fonction :

```
lieu *f(char *c) {
    lieu *premier;
    lieu *prec;
    lieu *current;
    for(premier = NULL; *c; c++) {
        if (premier == NULL) {
            premier = malloc(sizeof(lieu));
            if (!premier)
                return NULL;
            premier->suivant = NULL;
            premier->nom = c; //Met adresse du debut du nom
            current = premier;
        }
        if (*c == ' ') { //Prochain elem -> doit allouer
            prec = current;
            lieu *n;
            n = malloc(sizeof(lieu));
            if (!n) {
                libere(premier);
                return NULL;
            }
            current = n;
            current->suivant = premier;
            current->nom = c+1;
            prec->suivant = current;
```

```
        }
    }
    return premier;
}
void libere(lieu *p) {
    if (!p)
        return;
    lieu *premier = p;
    for (lieu *c = p->suivant; premier != p->suivant; p = suivant) {
        lieu *suivant = p->suivant;
        free(p);
    }
    if (p != p->suivant)
        free(p);
    free(premier);
}
```

(c) Cette fonction peut s'ecrire :

```
int f(lieu *p) {
    if (!p)
        return 0;
    unsigned n = 0;
    lieu *premier = p
    for (lieu *c = p->suivant; c != p->suivant; p = suivant) {
        lieu *suivant = p->suivant;
        free(p)
        n++;
    }
    return n+1;
}
```

4. (a) Le type structuré peut s'écrire :

```
typedef struct elem_t {
    int val;
    struct elem_t *next;
    struct elem_t *prev;
} elem;
```

(b) Cette fonction peut s'écrire :

```
elem *f (elem *p, int v, int i) {
    elem *n;
    n = malloc(sizeof(elem));
    n->val = v;
    n->prev = NULL;
    n->suiv = NULL;
    if (!p)
        return n;
    for (; p->prev; p = prev) { // Retourne au debut
        elem *prev = p->prev;
    }
    for (int j = 0; j < i; j++) {
```

```
            if (!p->suiv) {
                p->suiv = n;
                n->prev = p;
                return n;
            }
            p = p->suiv;
        }
        elem *suiv = p->suiv;
        p->suiv = n;
        n->prev = p;
        suiv->prev = n;
        n->suiv = suiv;

        return n;
    }
```

(c) Cette procédure peut s'écrire :

```
    void f(elem *p, int i) {
        if (!p)
            return;
        for (; p->prev; p = prev) {
            elem *prev = p->prev;
        }
        for (int j = 0; j < i; j++) {
            if (!p->suiv) {
                p->prev.suiv = NULL;
                free(p);
                return;
            }
            p = p->suiv;
        }
        p->prev.suiv = p->suiv;
        p->suiv.prev = p->prev;
        free(p);
    }
```