Correction des exercices de la session 6

1. Le type structuré peut s'écrire :

```
typedef struct page_t {
    char *url;
    unsigned long date;
    struct page_t *prec;
    struct page_t *suiv;
} page;
```

2. Nous allons réaliser cette fonction de deux façons différentes. La première en allouant séparément chaque page, la deuxième en allouant toutes les pages sous la forme d'un tableau.

```
page *f (char **url, unsigned long *temps, unsigned n) {
    web *premier, *current, *prev;
    premier = malloc(sizeof(page));
    if (!premier)
        return NULL;
    premier->url = url[0];
    premier->date = temps[0];
    premier->prec = NULL;
    premier->suiv = NULL;
    prev = premier;
    for (unsigned i = 1; i < n; i++) {
        current = malloc(sizeof(page))
        if (!current) {
            while (premier) {
                page *suiv = premier->suiv;
                free(premier);
                premier = suiv;
            }
            return NULL;
        }
        current->url = url[i];
        current->date = temps[i];
        current->prec = prev;
        prev->suiv = current;
        current->suiv = NULL;
        prev = current;
    }
    return premier;
}
```

```
page *f (char **url, unsigned long *temps, unsigned n) {
    page *web;
    web = malloc(sizeof(page)*n);
    if (!web)
        return NULL;
    for (unsigned i = 0; i < n; i++) {
        if (i == 0)
            web[i].prec = NULL;
        else
```

```
            web[i].prec = &web[i-1];
        if (i == n-1)
            web[i].suiv = NULL;
        else
            web[i].suiv = &web[i+1];
        web[i].url = url[i];
        web[i].date = temps[i];
    }
    return web;
}
```

3. Il faudra d'abord trouver la toute première page et ceci se fait de la même façon pour une liste liée ou un vecteur. Pour arriver à la fin, comme on ne connait pas la taille de la liste on devra passer par les pointeurs jusqu'à arriver à la fin :

```
unsigned long f (page *p) {
    while (p->prec) {
        p = p->prec;
    }
    unsigned long res = 0;
    while (p) {
        res += p->date;
        p = p->suiv;
    }
    return res;
}
```

4. Cette fonction sera différente en fonction de si on a utilisé un vecteur ou pas. La première version correspond à la première version de la fonction du point (2) et la deuxième à la deuxième version du point (2) :

```
void f (page *p) {
    while (p->prec) {
        p = p->prec;
    }
    while (p) {
        page *tmp = p->suiv;
        free(p);
        p = tmp;
    }
}

void f (page *p) {
    while (p->prec) {
        p = p->prec;
    }
    free(p);
}
```

5. Nous allons chaque fois faire chaque fonctions en deux versions. La première par un vecteur et la deuxième par une liste chainée.

   Commençons par définir les éléments :

```
typedef struct pile_t {
    int *elements;
    unsigned nb_elem;
    unsigned nb_max;
} pile;

typedef struct element_t {
    int valeur;
    struct element_t *suiv;
} element;
typedef struct pile_t {
    element *elements;
} pile
```

Pour la version avec un vecteur, on a besoin du nombre d'éléments qu'on peut placer car on va devoir allouer un vecteur d'éléments. Ce n'est pas le cas avec la version avec liste simplement chainée.

```
void push (pile *p, int v) {
    if (p->nb_elem +1 > p->nb_max) {
        printf("La pile est pleine, impossible d'ajouter une valeur\n");
        return;
    }
    p->valeur[p->nb_elem] = v;
    p->nb_elem++;
}

void push (pile *p, int v) {
    element *n;
    n = malloc(sizeof(element));
    if (!n) {
        printf("Erreur lors de l'allocation de l'element/n");
        return;
    }
    n->valeur = v;
    n->suiv = NULL;
    element *c = p->elements;
    if (!c)
        p->elements = n;
    else {
        while (c->suiv) {
            c = c->suiv;
        }
        c->suiv = n;
    }
}

void pop (pile *p) {
    if (p->nb_elem == 0) {
        printf("Pas d'elements a pop\n");
        return NULL;
    }
    p->nb_elem--;
    return p->elements[p->nb_elem];
```

```
}

element *pop (pile *p) {
    if (!p->elements) {
        printf("Pas d'elements a pop\n");
        return NULL;
    }
    element *e, *prev;
    prev = p->elements;
    if (!prev->suiv) {
        p->elements = NULL;
        return prev;
    }
    e = prev->suiv;
    while (e->suiv) {
        prev = e;
        e = e->suiv;
    }
    prev->suiv = NULL;
    return e;
}

element *top (pile *p) {
    if (!p->nb_elem)
        return NULL;
    return p->elements[p->nb_elem-1];
}

element *top (pile *p) {
    element *e = p->elements;
    if (!e)
        return NULL;
    while (e->suiv)
        e = e->suiv;
    return e;
}

unsigned top (pile *p) {
    return p->nb_elem;
}

unsigned top (pile *p) {
    element *e = p->elements;
    unsigned res = 0;
    while (e) {
        res++;
        e = e->suiv;
    }
    return res;
}
```

```
unsigned is_empty (pile *p) {
    if (p->nb_elem > 0)
        return 1;
    return 0;
}

unsigned is_empty (pile *p) {
    if (p->elements)
        return 1;
    return 0;
}
```