

Examen de Langage C

Durée : 1h30

Aucun document autorisé – Mobiles interdits

Notez que les affirmations (antécédents, conséquents, rôles, et invariants) dans vos codes C entreront pour partie dans la note finale.

- 1. En utilisant uniquement la notation pointeur, et SANS UTILISER de fonctions de `string.h`, réécrivez la fonction `strrchr` qui renvoie un pointeur sur la dernière occurrence d'un caractère `c` dans une chaîne de caractères `s`. Si le caractère `c` n'est pas dans `s`, la fonction renvoie `NULL`. La chaîne et le caractère sont les deux paramètres de cette fonction.

```
/**
 * Rôle : renvoie un pointeur sur la dernière occurrence du caractère
 * c dans la chaîne s. Renvoie NULL si c ∉ s
 */
const char *strrchr(const char *s, const char c) {
    const char *p = NULL;
    while (*s) {
        if (*s == c) p = s;
        s++;
    }
    return p;
}
```

ou bien

```
/**
 * Rôle : renvoie un pointeur sur la dernière occurrence du caractère
 * c dans la chaîne s. Renvoie NULL si c ∉ s
 */
const char *strrchr(const char *s, const char c) {
    const char *p = s;
    while (*s++);
    while (--s >= p)
        if (*s == c) return s;
    // c ∉ s
    return NULL;
}
```

- 2. En utilisant uniquement la notation pointeur, SANS UTILISER de fonctions de `string.h`, écrivez la fonction `stpcpy`. Comme `strcpy`, cette fonction copie une chaîne de caractères source `src` vers une chaîne de caractères destination `dest`, mais s'assure de ne pas dépasser une adresse de fin `destend`. Comme `strcpy`, `stpcpy` renvoie le pointeur `dest`. Cette fonction possède l'en-tête suivant :

/*

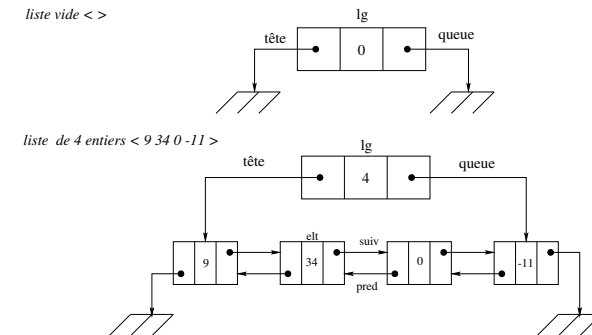
```
* Rôle : .....
*/
char *stpcpy(char *dest, const char *destend, const char *src)

.....

/*
 * Rôle : copie src dans dest sans dépasser l'adresse destend
 *
 */
char *stpcpy(char *dest, const char *destend, const char *src) {
    char *p = dest;
    while (dest < destend)
        if ((*dest++ = *src++) == '\0')
            // toute la chaîne src a été copiée
            return p;
    // dest >= destend => limite atteinte
    *dest = '\0';
    return p;
}
```

On souhaite mettre en œuvre la notion de *liste linéaire* (telle que vue en TD) à l'aide d'une structure dynamique doublement chaînée. Les éléments de la liste sont placés dans nœuds reliés par des pointeurs.

Le schéma donné ci-dessous, montre, par exemple, l'organisation d'une liste vide `<>`, et de la liste d'entiers `< 9, 34, 0, 11 >`.



- 3. À partir du schéma précédent, déclarez les types `liste`, `noeud`, et `pnoeud` (pointeur sur `noeud`) du fichier `liste.h`. Dans la suite de ce DS, vous utiliserez ces déclarations de type.

```
#pragma once;
typedef int T;
typedef ... pnoeud;
typedef ... liste;
```

```
typedef struct noeud {
    T elt;
    struct noeud *pred, *suiv;
} *pnoeud;
```

```
typedef struct {
    int lg;
    pnoeud tête, queue;
} liste;
```

.....

- 4. Écrivez les fonctions `listeVide` et `longueur` qui renvoient, respectivement, une liste vide et la longueur d'une liste. Ces fonctions ont les en-têtes suivants :

`liste listeVide(void)` et `int longueur(const liste l)`.

```
/*
 * Rôle : renvoie une liste vide
 */
liste listeVide(void) {
    return (liste) {0, NULL, NULL};
}

/*
 * Rôle : renvoie la longueur de la liste l
 */
int longueur(const liste l) {
    return l.lg;
}

.....
```

- 5. Écrivez la fonction `ième` qui renvoie l'élément de rang `r` d'une liste `l`. Le rang est compris entre 1 et `longueur(l)`.

```
/*
 * Antécédent :  $1 \leq r \leq \text{longueur}(l)$ 
 * Rôle : renvoie l'élément de rang r de la liste l
 */
T ième(const liste l, const int r) {
    assert(r>=1 && r<=longueur(l));
    pnoeud p = l.tête;
    for (int i=1; i<r; i++)
        p = p->suiv;
    // p désigne l'élément de rang r
    return p->elt;
}

.....
```

- 6. Écrivez la procédure `supprimer` qui supprime un élément au rang `r` dans une liste `l` de type `liste`.

```
/*
 * Antécédent :  $1 \leq r \leq \text{longueur}(l)+1$ 
 * Rôle : insère l'élément x au rang r dans la liste l
 */
void supprimer(liste *l, const int r) {
    assert(r>=1 && r<= l->lg);
    pnoeud q;
    if (l->lg == 1) {
        // un seul élément => r == 1
        q = l->tête;
        l->tête = l->queue = NULL;
    }
    else // au moins 2 éléments
        if (r == 1) { // suppression en tête de liste
            q = l->tête;
            l->tête = q->suiv;
            l->tête->pred = NULL;
        }
        else
            if (r==l->lg) {
                // suppression du dernier élément de la liste
                q = l->queue;
                l->queue = l->queue->pred;
                l->queue->suiv = NULL;
            }
            else { // cas général,  $r > 1$  et  $r < \text{lg}$ 
                pnoeud p = NULL;
                q = l->tête;
                for (int i=1; i<r; i++) {
                    p = q;
                    q = q->suiv;
                }
                // q désigne l'élément de rang r
                q->suiv->pred=p;
                p->suiv = q->suiv;
            }
    //
    l->lg--;
    free(q);
}
```

.....

- 7. Écrivez la procédure `ajouterEnQueue` qui ajoute un élément `x` à la fin d'une liste passée en paramètre. Évidemment, il ne faut pas parcourir la liste.

```
/*
 * Rôle : renvoie le pointeur sur noeud créé et initialisé à x
 */
static pnoeud créerNoeud(const T x) {
    pnoeud p = malloc(sizeof(struct noeud));
    p->elt = x;
    p->pred = p->suiv = NULL;
    return p;
}
```

```

/*
 * Rôle : ajoute en fin de liste l l'élément x
 */
void ajouterEnQueue(liste *l, const T x) {
    pnoeud n = créerNoeud(x);
    if (l->lg==0)
        // liste vide
        l->tête = l->queue = n;
    else {
        // ajouter en queue
        n->pred = l->queue;
        l->queue->suiv = n;
        l->queue = n;
    }
    // incrémenter de 1 le nb d'éléments
    l->lg++;
}

```

.....

- 8. Écrivez la fonction `main` qui prend en paramètre programme le nom d'un fichier d'entiers (type `int`, attention ce n'est pas un fichier de texte), et qui crée (à l'aide exclusivement des fonctions et procédures précédentes) une liste (type `liste`) formée de tous les entiers contenus dans le fichier. Vous ferez les vérifications nécessaires.

```

int main(int argc, char *argv[]) {
    if (argc!=2) {
        fprintf(stderr, "Usage : %s file\n", argv[0]);
        return EXIT_FAILURE;
    }
    // argc valide
    FILE *fp;
    if ((fp = fopen(argv[1], "r"))==NULL) {
        perror(argv[1]);
        return errno;
    }
    // argv[1] ouvert en lecture => fabriquer la liste d'entiers
    liste l = listeVide();
    int x;
    while (fread(&x, sizeof(int), 1, fp)>0)
        ajouterEnQueue(&l, x);
    // EOF => fermer le fichier
    if (fclose(fp)==EOF) {
        perror(argv[1]);
        return errno;
    }
    return EXIT_SUCCESS;
}

```

.....