

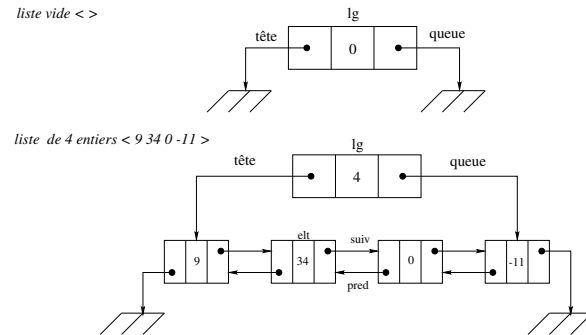
Examen de Info C

Durée : 1h30

Aucun document autorisé
Mobiles interdits

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 *noeuds* reliés par des pointeurs.

Le schéma donné ci-dessous, montre, par exemple, l'organisation d'une liste vide $\langle \rangle$, et de la liste d'entiers $\langle 9, 34, 0, 11 \rangle$.



- 1. 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;
```

- 2. 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;
}
```

- 3. 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 ≤ r ≤ 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;
}
```

- 4. Écrivez la procédure `supprimer` qui supprime un élément au rang `r` dans une liste `l` de type `liste`. Le rang est compris entre 1 et `longueur(l)`.

```
/*
 * Antécédent : 1 ≤ r ≤ longueur(l)
 * Rôle : supprime 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 < 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);
}

```

- 5. 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;

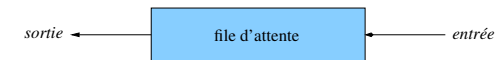
```

```

else {
// ajouter en queue
n->pred = l->queue;
l->queue->suiv = n;
l->queue = n;
}
// incrémenter de 1 le nb d'éléments
l->lg++;
}

```

Les `files` définissent le modèle *premier entré – premier sorti* (en anglais FIFO - *First-In First-Out*). Les éléments sont insérés dans la séquence par une des extrémités et en sont extraits par l'autre. Ce modèle correspond à la file d'attente que l'on rencontre bien souvent face à un guichet dans les bureaux de poste, ou à une caisse de supermarché la veille d'un week-end. À tout moment, seul le premier client de la file accède au guichet ou à la caisse.



Sur une file de type `file`, on définit les fonctions et procédures suivantes :

1. `file fileVide(void)` //renvoie une file vide
 2. `bool estVide(const file f)` //teste si la file vide est vide ou non
 3. `T premier(const file f)` //renvoie l'élément en tête de file
 4. `void enfiler(file *f, const T x)` //ajoute un élément en queue de file
 5. `void défiler(file *f)` //supprime l'élément en tête file
- 6. Dans la mesure où une file *est* une liste particulière, on décide de mettre en œuvre une file à l'aide d'une liste. Avec `typedef`, déclarez le type `file`.

```
typedef liste file;
```

- 7. À l'aide des fonctions et procédures de manipulation d'une liste que vous avez écrites précédemment, écrivez les 5 fonctions et procédures de manipulation de file d'attente données au-dessus.

```

/*
 * Rôle : renvoie une file vide
 */
file fileVide(void) {
return listeVide();
}

/*
 * Rôle : teste si la file est vide ou non
 */
bool estVide(const file f) {
return longueur(f)==0;
}

```

```

}

/*
 * Antécédent : f non vide
 * Rôle : renvoie l'élément en tête de file
 */
T premier(const file f) {
    return ième(f, 1);
}

/*
 * Rôle : ajoute l'élément x en queue de file
 */
void enfiler(file *f, const T x) {
    ajouterEnQueue(f, x);
}

/*
 * Antécédent : f non vide
 * Rôle : supprime le 1er élément de la file
 */
void défiler(file *f) {
    supprimer(f, 1);
}

```

La déclaration du type T de la première page ne permet de manipuler que des listes ou des files d'entiers. Pour manipuler des listes et des files *génériques*, dont les éléments sont de type quelconque, on re-déclare T de la façon suivante : **typedef void * T;**

- 8. Dans une fonction main :
- déclarez et initialisez une file générique f;
 - enfiler successivement la chaîne de caractères "bonjour" et les réels **double** 12.99 et -3.14;
 - affichez le premier élément de la file f et défilez f;
 - affichez le premier élément de la file f et défilez f;
- Vous pouvez écrire des fonctions auxiliaires.

```

/*
 * Rôle : alloue dynamiquement le réel double d
 *         et renvoie le pointeur qui le désigne
 */
double *makeDouble(const double d) {
    double *p = malloc(sizeof(double));
    *p = d;
    return p;
}

/*
 * Rôle : renvoie le réel double pointé par d
 */
double getDouble(const void *d) {
    return *(double *) d;
}

int main(int argc, char *argv[]) {

```

```

file f = fileVide();

enfiler(&f, "bonjour");
enfiler(&f, makeDouble(12.99));
enfiler(&f, makeDouble(3.14));

printf("%s\n", (char *) premier(f)); défiler(&f);
printf("%.2f\n", getDouble(premier(f))); défiler(&f);

return EXIT_SUCCESS;
}

```

Un fichier (**PAS UN FICHIER DE TEXTE**) contient une suite de réels **double**. La suite est de longueur quelconque, éventuellement vide.

- 9. Écrivez la fonction fileDeDoubles qui lit un fichier de réels **double** et renvoie une file d'attente contenant les réels **double** contenus dans fichier. Les éléments de la file d'attente restent déclarés de type **void ***. L'en-tête de la fonction est le suivant :

```
file fileDeDoubles(const char *nomf)
```

```

file fileDeDoubles(const char *nomf) {
    FILE *fd;
    if ((fd=fopen(nomf, "r"))==NULL) {
        perror(nomf);
        exit(errno);
    }
    // le fichier est ouvert en lecture
    file f = fileVide();
    double d;
    while (fread(&d, sizeof(double), 1, fd)>0)
        enfiler(&f, makeDouble(d));
    //
    fclose(fd);
    return f;
}

```

- 10. Écrivez la fonction main qui utilise votre fonction fileDeDoubles pour créer une file d'attente de **double**, puis, qui affiche sur la S/S tous les réels de la file d'attente. Le nom du fichier est donné par argv[1]. 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;
    }
    file f = fileDeDoubles(argv[1]);
    // afficher le contenu de la file d'attente
    while (!estVide(f)) {

```

```
    printf("%0.3lf ", getDouble(premier(f)));  
    défiler(&f);  
}  
printf("\n");  
  
return EXIT_SUCCESS;  
}
```

.....