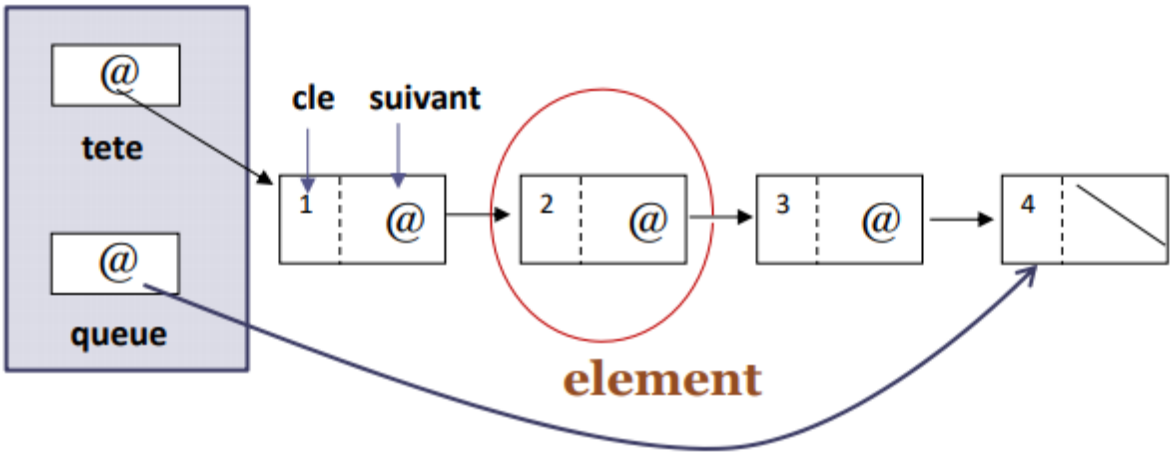
LES FILES

File Répresentation chainée

File	:	Cle Tete **//(sommet)** queue



La SD file d'attente est une SD à deux points d'entrée
La SD pile est une SD à un seul point d'entrée

FileRCHTDA.h

```c
/* représentation Chainée */
struct element
{
    int cle;
    struct element *suivant;
};
struct file
{
    struct element *tete;
    struct element *queue;
};

/*opération ou services exportés*/
struct file *creer_file () ;
unsigned file_vide ( struct file * ) ;
int premier (struct file * ) ;
void enfiler (int, struct file *) ;
void defiler (struct file *) ;
```

FileRCHTDA.c

```c
#include <alloc.h>
#include <assert.h>
#include "FileRCHTDA.h"

struct file *creer_file()
{
    struct file *f;
    f = (struct file *)malloc(sizeof(struct file));
    f->tete = NULL;
    f->queue = NULL;
    return f;
}

unsigned file_vide(struct file *f)
{
    return(f->tete == NULL && f->queue == NULL);
}
```

```
int premier(struct file *)
{
    assert(!file_vide(f));
    return (f->tete->cle);
}

void enfiler(int info, struct file *f)
{
    // preparation du nouvel élément
    struct element *p;
    p= (struct element *) malloc(sizeof(struct element);
    p->cle= info ;
    p->suivant = NULL;
    // chainage : deux cas  à envisager
    if (file_vide(f)
    {
        // cas file vide
        f->tete = p;
    }
    Else
    {
        // cas ou la file n'est pas vide
        f->queue->suivant = p;
    }
    f->queue = p;
}

void defiler(struct file *)
{
    struct element *p;
    assert(!file_vide(f));
    P = f->tete;
    f->tete = f->tete->suivant;
    free(p);
    if (f->tete == NULL)
        f-
>queue = NULL; // cas où la file contenait 1 seul élément avant le défilement
}
```

FileRCHOA.h

```
/*opération ou services exportés*/
void creer_file ();
unsigned file_vide ();
int premier ();
void enfiler (int);
void defiler ();
```

FileRCHOA.c

```
#include <alloc.h>
#include <assert.h>
#include "FileRCHTDA.h"


/* représentation Chainée */
struct element
{
    int cle;
    struct element *suivant;
};
static struct element *tete;
static struct element *queue;


struct file creer_file()
{
    tete = NULL;
    queue = NULL;
}

unsigned file_vide()
{
    return(tete == NULL && queue == NULL);
}

int premier()
{
    assert(!file_vide());
    return (tete->cle);
```

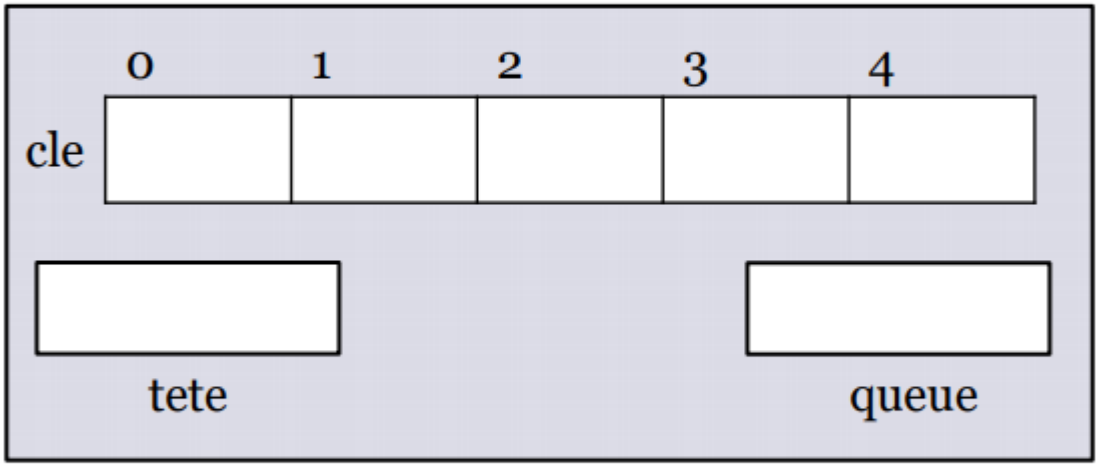
```
}

void enfiler(int info)
{
    // preparation du nouvel élément
    struct element *p;
    p= (struct element *) malloc(sizeof(struct element);
    p->cle= info ;
    p->suivant = NULL;
    // chainage : deux cas  à envisager
    if (file_vide(f)
    {
        // cas file vide
        tete = p;
    }
    Else
    {
        // cas ou la file n'est pas vide
        queue->suivant = p;
    }
    queue = p;
}

void defiler()
{
    struct element *p;
    assert(!file_vide(f));
    p = tete;
    tete = tete->suivant;
    free(p);
    if (tete == NULL)
        queue = NULL; // cas où la file contenait 1 seul élément avant le défilem
ent
}
```

File Répresentation contingue

Illustration



tete est un indice correspondant au premier élément de la file
queue est un indice correspondant au dernier élément de la file

Matérialisation de la SD file d'attente

creer_file
- procédure : void creer_file(struct file *);
- fonction : struct file * créer_file(void); /* paramètre out */

On a intérêt à faire un passage par adresse dans le cas de la représentation physique contigüe, car dans le cas d'un passage par valeur on doit déplacer tous les éléments que contient la file
Lorsqu'on fait un passage par adresse on ne fait déplacer à l'environnement appelé que l'adresse de la file

La passage par valeur cause une perte de temps et d'espace

FileRCOTDA.h

```c
/* représentation contigüe */
#define n 100
struct file
{   int cle[n] ;
    unsigned tete;
    unsigned queue;
};

/*opération ou services exportés*/
void creer_file(struct file*) ;
    /* struct file* creer_file(void) ;*/
unsigned file_vide(struct file) ;
    /* unsigned file_vide(struct file *) ; */
int premier(struct file);
    /* int premier(struct file *);*/
void enfiler(int, struct file*);
void defiler(struct file*);
```

FileRCOTDA.c

```c
#include<assert.h>
#include "FileROTDA.h"

void creer_file(struct file*f)
{
    f->tete=0 ;
    f->queue=0 ;
}
//Remarque : n'importe quel indice compris entre 0 et n-
1 pourrai  être considéré

unsigned file vide(struct file f)
{
    return(f.tete == f.queue);
}

int premier(struct file f)
{
    unsigned i;
    assert(!file vide(f));
    i=f.tete+1 ;
    if(i>n-1)
```

```
        i=0;
    return(f.cle[i]);
}

void enfiler(int info,struct file*f)
{
    f->queue++;
    if(f->queue>n-1)
    f->queue=0;
    assert(f->tete!=f->queue) ;
    /* f->tete==f->queue correspond  ici à une file pleine */
    f->cle[f->queue]=info ;
}

void defiler(struct file *f)
{
    assert(!file vide(*f));
    f->tete++ ;
    if(f->tete>n-1)
    f->tete=0 ;
}
```

1	5	6	7	8
0	1	2	3	4

Creation

Tete → 0

Queue → 0

Enfiler(5)

Queue --> 1 (premier il est dans 1 qui est (i+1))

Enfiler (6)

Queue → 2

Enfiler(7)

Queue → 3

Defiler()

Tete →1 (premier est dans 2)

Queue reste 3

Enfiler (8)

Queue → 4

Enfiler (1)

Queue →5 puis queue →0

FileRCOOA.h

```
/*opération ou services exportés*/
void creer_file() ;
unsigned file_vide() ;
int premier();
void enfiler(int);
void defiler();
```

FileRCOOA.c

```
#include<assert.h>
#include "FileROTDA.h"


/* représentation contigüe */
#define n 100
struct file
{   int cle[n] ;
    unsigned tete;
    unsigned queue;
};
static struct file f;

void creer_file()
{
    f.tete=0 ;
    f.queue=0 ;
}
//Remarque : n'importe quel indice compris entre 0 et n-1 pourrai  être considéré

unsigned file vide()
{
    return(f.tete == f.queue);
}

int premier()
{
    unsigned i;
    assert(!file vide());
    i=f.tete+1 ;
    if(i>n-1)
        i=0;
```

```
    return(f.cle[i]);
}

void enfiler(int info)
{
    f.queue++;
    if(f.queue > n-1)
    f.queue=0;
    assert(f.tete!=f.queue) ;
    /* f.tete==f.queue correspond  ici à une file pleine */
    f.cle[f.queue]=info ;
}

void defiler()
{
    assert(!file vide());
    f.tete++ ;
    if(f.tete>n-1)
    f.tete=0 ;
}
```

Matérialisation de la SD file d'attente

Discussion

- Question
  - En partant de cette file vide, exécuter la séquence suivante
    a) enfiler les éléments : 50, 20, 30, 40, 13
    b) defiler
    c) enfiler 120
  - Résultat de l'exécution

Convention :
On enfile après queue (queue change)
On défile après tete (tete change)

a) Situation, initiale tete=0 queue=0
Situation finale tete=0 queue=5

b) Situation initiale tete=0 queue=5
Situation finale tete=1 queue=5

c) L'état en position 0 est disponible ??

Discussion

- Problème
  - On ne peut enfiler 120 après queue
  - en effet l'élément de position queue (5) n'appartient pas au tableau cle
  - pourtant la file n'est pas pleine
  - car le tableau est perçu d'une façon linéaire, il est parcourue de gauche à droite

Au bout de n enfilements, on ne peut plus ajouter des nouveaux éléments, même si on fait des défilements avec n est la taille du tableau

Matérialisation de la SD file d'attente

Solution 1

- Un tableau circulaire
  - tete et queue varient modulo n avec n est la taille du tableau
- Il s'agit d'une perception **logique** et **non physique**
- On va appliquer la séquence des actions a, b, c vue précédemment sur un tableau perçu d'une façon circulaire

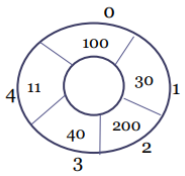
Solution 1

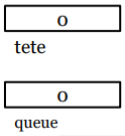
a) enfilement 100 queue=1
enfilement 20 queue=2
enfilement 30 queue=3
enfilement 40 queue=4
enfilement 13 queue=?
queue+1>n-1 (5>4) ➔ queue=0

b) defiler tete=1
c) enfiler 120
queue+1=0+1=1
cette position est disponible

Matérialisation de la SD file d'attente

Discussion





La file est pleine ???

- tete=queue
  - file vide ou file pleine

Le problème c'est que il s'agit d'une proposition ambiguë

Solution 2

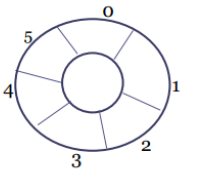
Au lieu de réserver un tableau (cle) de n éléments, on prévoit un tableau de taille N=n+1 en respectant la propriété suivante

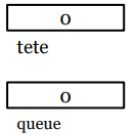
On utilise au plus n éléments s lorsque la file contient n éléments, la file est logiquement pleine mais pas physiquement

La proposition tete=queue caractérise une file vide

Matérialisation de la SD file d'attente

Solution 2 : Illustration

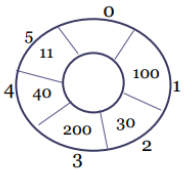


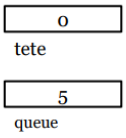


file vide

Solution 2 : Illustration

- On réserve 6 (5+1) éléments
- On utilise au maximum 5 éléments





file pleine

Logiquement pleine mais pas physiquement