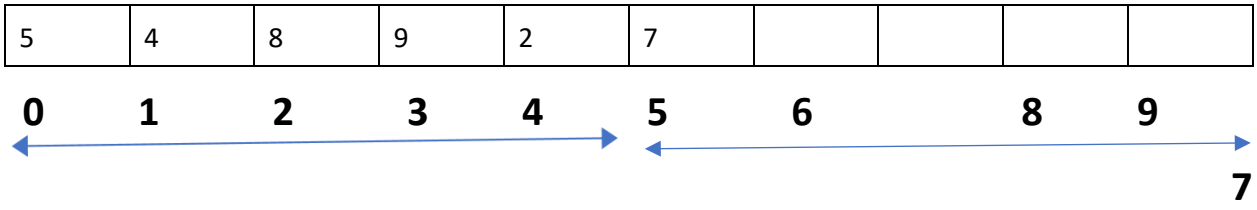
Pile

Pile Representation Contigue :

En tant que OA :

{

Cle[10] Sommet = 4



Partie utilisé

Partie non utilisée

}

```
dernier
return(p.cle[p.sommet]);

empiler
p.sommet++;
p.cle[p.sommet]=info;

depiler
assert(!vide());
p.sommet--;
```

PileRCOOA.h

```
/* Prototypes des fonctions */
/* Operation de creation */
struct element creer_pile();

/* Operation de consultation */
unsigned vide();
int dernier();

/* Operation de modification */
void empiler(int);
void depiler();
```

PileRCOOA.c

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

/* DÃ©finition d'une Pile */
#define N1 100
struct pile
{
    int cle[N];
    int sommet;
};
static struct pile p;
/* Operation de creation */
struct element creer_pile()
{
    p.sommet=-1;
}

/* Operation de consultation */
unsigned vide()
{
    return(p.sommet==-1);
}
```

```
int dernier()
{
    assert(!vide());
    return(p.cle[p.sommet]);
}

/* Operation de modification */
void empiler(int info)
{
    p.sommet++;
    p.cle[p.sommet]=info;
}

void depiler()
{
    assert(!vide());
    p.sommet--;
}
```

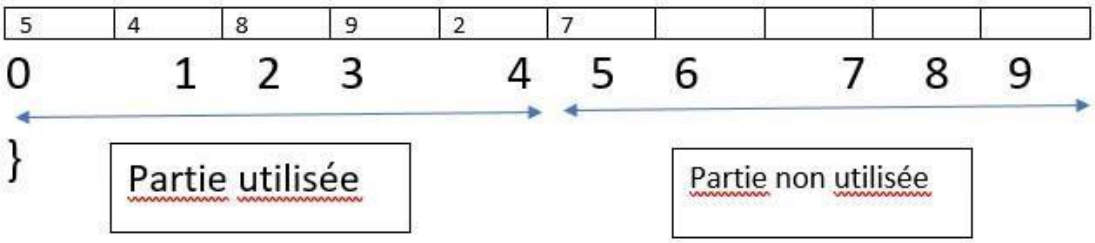
En tant que TDA :

Pile p

{

Cle[10]

Sommet = 4



}

Partie utilisée

Partie non utilisée

PileRCOTDA.h

```
/* DÃ©finition d'une Pile */

#define N1 100
struct pile
{
    int cle[N];
    int sommet;
};

/* Prototypes des fonctions */
/* Operation de creation */
struct element *creer_pile(struct pile*);

/* Operation de consultation */
unsigned vide(struct pile);
int dernier(struct pile);

/* Operation de modification */
void empiler(int,struct pile*);
void depiler(struct pile*);
```

PileRCOTDA.c

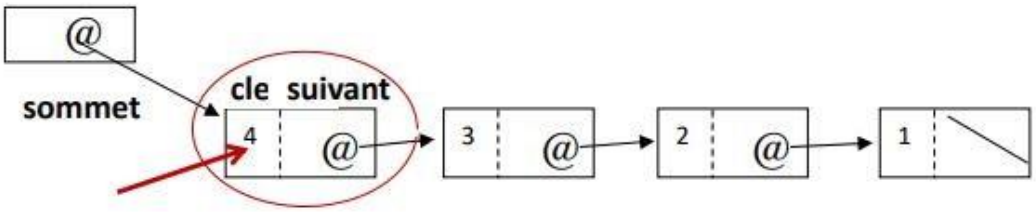
```c
#include<assert.h>

#include "PileOA.h"

/* Operation de creation */
struct element creer_pile(struct pile *p)
{
    p->sommet=-1;
}

/* Operation de consultation */
unsigned vide(struct pile p)
{
    return(p.sommet==-1);
}

int dernier(struct pile p)
{
    assert(!vide(p));
    return(p.cle[p.sommet]);
}

/* Operation de modification */
void empiler(int info,struct pile *p)
{
    p->sommet++;
    p->cle[p->sommet]=info;
}

void depiler(struct pile *p)
{
    assert(!vide(p));
    p->sommet--;
}
```

Pile Representation Chainée :

En tant que OA :

Représentation Abstraite d'une pile vide

Représentation Chaînée d'une pile vide



PileRCHOA.h

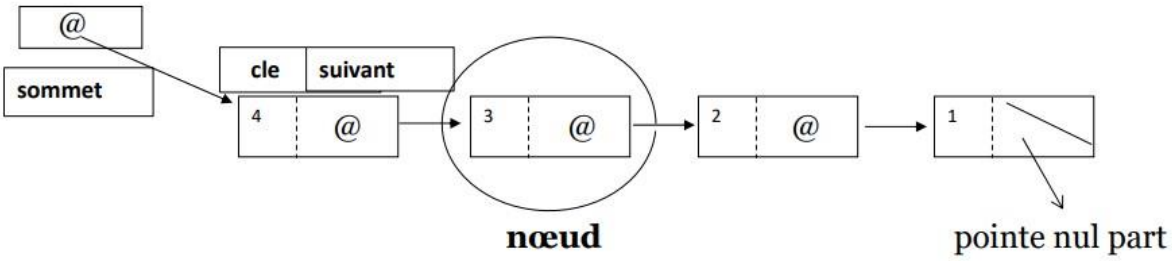
```c
/* Prototypes des fonctions */

/* Operation de creation */
struct element *creer_pile(void);

/* Operation de consultation */
unsigned vide(void); //1 si la pile est vide sinon 0
int dernier(void);

/* Operation de modification */
void empiler(int);
void depiler(void);
```

PileRCHOA.c

```c
#include<stdio.h>
#include<assert.h>
#include<stdlib.h>
#include "PileRCHOA.h"

/* DÃ©finition d'une Pile */

struct element
{
    int cle;
    struct element *suivant;
};
struct element *sommet;
/* Operation de creation */
struct element creer_pile(void)
{
    sommet=NULL;
}

/* Operation de consultation */
unsigned vide(void)
{
    //1 si la pile est vide sinon 0
    return(sommet==NULL);
}

int dernier(void)
{
    assert(!vide());
    return(sommet->cle);
}

/* Operation de modification */
void empiler(int info)
{
    struct element *p;
    p=(struct element*)malloc(sizeof(struct element*));
    p->cle=info;
    p->suivant=sommet;
    sommet=p;
    nb++;
}
```

```c
void depiler(void)
{
    struct element *p;
    assert(!vide());
    p=sommet;
    sommet=sommet->suivant;
    free(p);
    nb--;
}
```

En tant que TDA :



PileRCHTDA.h

```
 /* DÃ©finition d'une Pile */


struct element
{
    int cle;
    struct element *suivant;
};

/* Prototypes des fonctions */
/* Operation de creation */
struct element *creer_pile(void);

/* Operation de consultation */
unsigned vide(struct element*); //1 si la pile est vide sinon 0
int dernier(struct element*);

/* Operation de modification */
void empiler(int,struct element**);
void depiler(struct element**);
```

PileRCHTDA.c

```c
#include<stdio.h>
#include<assert.h>
#include<alloc.h>
#include "PileRCHTDA.h"

/* Operation de creation */
struct element creer_pile(void)
{
    return NULL;
}

/* Operation de consultation */
unsigned vide(struct element *p)
{
    //1 si la pile est vide sinon 0
    return(p==null);
}

int dernier(struct element *p)
{
    assert(!vide(p));
    return(p->cle);     // return((*sommet).cle)
}

/* Operation de modification */
void empiler(int info,struct element* *p)
{
    struct element *q;
    q=(struct element*)malloc(sizeof(struct element*));
    q->cle=info;
    q->suivant=*p;
    *p=q;
}

void depiler(struct element* *p)
{
    struct element *q;
    assert(!vide(*p));
    q=*p;
    *p=q->suivant;
    free(q);
}
```