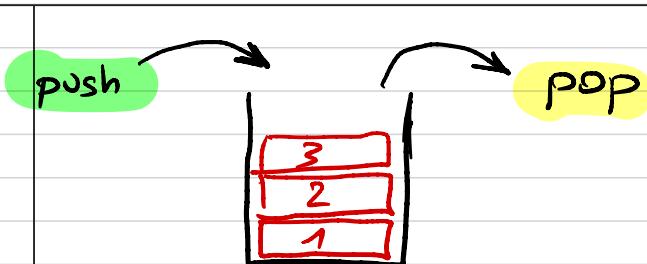


Listes : PILES

3. VI. 2021



Push: ajoute un élément sur la pile

Pop :

- 1) Lit l'élément sur la pile
- 2) On le supprime de la pile

Comment utiliser la liste tableau pour faire une pile

[0] [1] [2]



push: ajoute l'élément en fin de list:

`insertElemAt (list, END_OF_LIST, e)`

pop:

- Lit le dernier élément à la liste
- Supprime le dernier élément à la liste

`e → e [numElem - 1]`

`deleteElemAt (list, END_OF_LIST)`

① TD 2021/06/03
② ajouter les

(copie du TD 2021/05/15)
2 fonctions

- `push (list, e)`
- `e = pop (list)`

15h 10

③ Test dans le main :

`push (1)`
`push (2)`
`push (3)`

<code>e = pop</code>	\rightarrow	3
<code>e = pop</code>	\rightarrow	2
<code>e = pop</code>	\rightarrow	1

→ Exemple d'implémentation d'un pilo "LIFO"

```
// lifo.h

#pragma once

#include "list.h"
#include "error.h"

eErrorCode push(pList l, Element e);
Element pop(pList l);
```

```
// lifo.c

#include "lifo.h"

eErrorCode push(pList l, Element e) {
    eErrorCode returnCode = E_NO_ERROR;

    if(l!=NULL) {
        insertElemAt(l, END_OF_LIST, e);
    }
    else {
        returnCode = E_BAD_LIST;
    }
    return returnCode;
}

Element pop(pList l) {
    Element e;
    if (l != NULL)
    {
        e = l->e[l->numElem - 1];
        deleteElemAt(l, END_OF_LIST);
    }
    else {
        printf("Error bad list.\n");
    }
    return e;
}
```

```
sList l;
Element e;

initList(&l);
push(&l, 1.0);
push(&l, 2.0);
push(&l, 3.0);
displayList(&l);
e = pop(&l);
printf("e=%lf\n", e);
displayList(&l);
e = pop(&l);
printf("e=%lf\n", e);
displayList(&l);
e = pop(&l);
printf("e=%lf\n", e);
freeList(&l);
```

[1] [2] [3] ← num_elem - 1

1 2 3

num_elem = 3

```
→ TD20210603 git:(main) ✘ ./app
1.000 2.000 3.000
e=3.000000
1.000 2.000
e=2.000000
1.000
e=1.000000
```

list : queue d'éléments

put



→ get

put : ajout d'un élément dans la queue

get : - lire l'élément depuis la queue
- Supprime l'élément

get



← put

put : insertion d'un élément en fin de liste

get : - lecture du 1^{er} élément de la liste
- suppression du 1^{er} élément

Implementation

TD 2021.06.03.

f; fo. c
f; fo. h

2 fonctions put et get

Et tout dans le main

16^h45

```
// fifo.h

#pragma once

#include "list.h"
#include "error.h"

eErrorCode put(pList l, Element e);
Element get(pList l);
```

```
// fifo.c

#include "fifo.h"

eErrorCode put(pList l, Element e) {
    eErrorCode returnCode = E_NO_ERROR;

    if(l!=NULL) {
        insertElemAt(l, END_OF_LIST, e);
    }
    else {
        returnCode = E_BAD_LIST;
    }
    return returnCode;
}

Element get(pList l) {
    Element e;
    if (l != NULL)
    {
        e = l->e[0]; 1st elem
        deleteElemAt(l, 0);
    }
    else {
        printf("Error bad list.\n");
    }
    return e;
}
```

```
sList l;
Element e;

initList(&l);
put(&l, 1.0); 1.000
put(&l, 2.0); 2.000
put(&l, 3.0); 3.000
displayList(&l);
e = get(&l); e=1.000000
printf("e=%lf\n", e);
displayList(&l);
e = get(&l); e=2.000000
printf("e=%lf\n", e);
displayList(&l);
e = get(&l); e=3.000000
printf("e=%lf\n", e);
freeList(&l);
```

```
→ TD20210603 git:(main) ✘ ./app
1.000 2.000 3.000
e=1.000000
2.000 3.000
e=2.000000
3.000
e=3.000000
```

listes chainées.

liste tableau

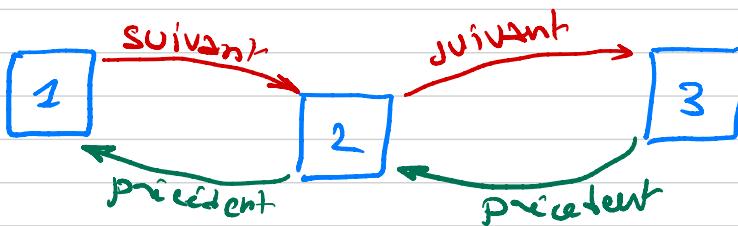


$t[\text{num}]$

zone non utilisée.

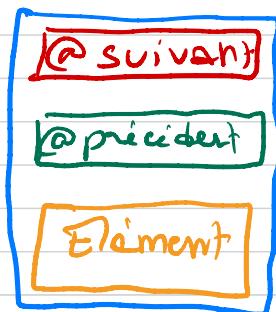
Taille limitée ou extensible (alloc dynamique).
Tsl du tableau : parcsse du tableau
échange entre case.

list. chaînée



- éléments alloués dynamiquement.
- on ajoute des liens entre les éléments pour former une chaîne

1 élément se compose de



1

3

2

typ. Element de notre liste tableau
fifO ou pile

Cette nouvelle structure "sElem" contient 3 champs

typedef struct sElem {

struct sElem *next;
struct sElem *prev;
Element;

} sElem;

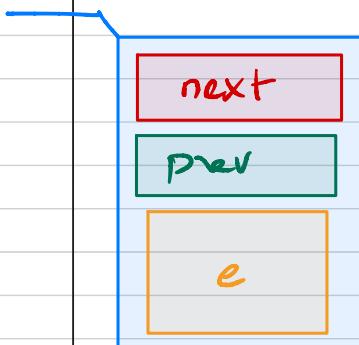
notation spéciale qui crée le type "struct sElem" que l'on peut utiliser pour les champs next et prev.

```

typedef struct sElem {
    struct sElem *next;
    struct sElem *prev;
    Element e;
} sElem;

```

sElem



① sElem *p = NULL;

$p = (\text{sElem} *) \text{malloc}(\text{sizeof}(\text{sElem}))$; ($p = 1000$)
 $p \rightarrow \text{next} = \text{NULL}$;
 $p \rightarrow \text{prev} = \text{NULL}$;
 $p \rightarrow e = 3.$;

②

$\text{sElem } *q = \text{NULL}$;
 $q = (\text{sElem} *) \text{malloc}(\text{sizeof}(\text{sElem}))$; ($q = 2000$)
 $q \rightarrow \text{next} = \text{NULL}$;
 $q \rightarrow \text{prev} = \text{NULL}$;
 $q \rightarrow e = 1.$;

③

$\text{sElem } *r = \text{NULL}$;
 $r = (\text{sElem} *) \text{malloc}(\text{sizeof}(\text{sElem}))$; ($r = 3000$)
 $r \rightarrow \text{next} = \text{NULL}$;
 $r \rightarrow \text{prev} = \text{NULL}$;
 $r \rightarrow e = 1.$;

a) $p \rightarrow \text{next} = q$

b) $q \rightarrow \text{next} = r$

c) $r \rightarrow \text{prev} = q$

d) $q \rightarrow \text{prev} = p$

