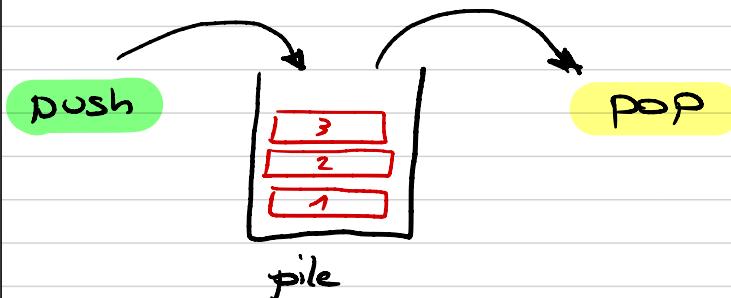


listes:

PILES

7.5.2021

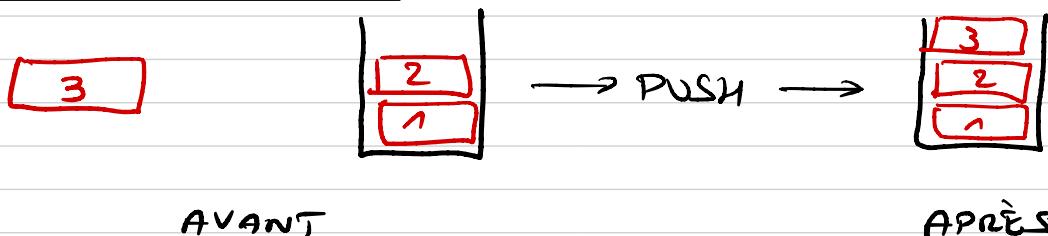
liste tableau, support pour d'autres types de liste.



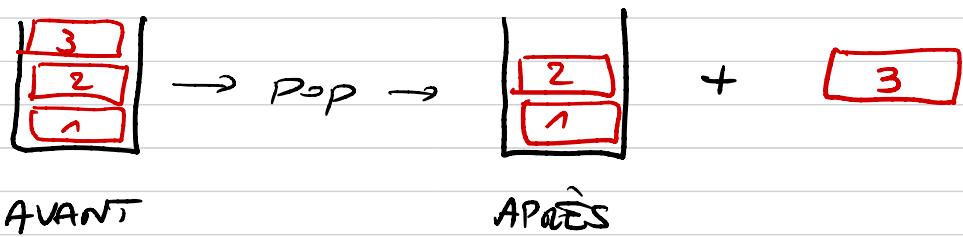
push: ajoute d'un élément sur la pile

pop: 1) lecture de l'élément au dessus de la pile
2) suppression de cet élément

Détail du push



Détail du Pop



Utilisation list.-> tableau.
(o) [e]



push: insertion de l'élément en fin de liste.

$c = \ell \rightarrow c [P \rightarrow \text{numElém} - 1]$

pop: lire le dernier élément
supprimer le dernier élément.

TD 2020/2021 → TD 2020/2021

2 fonctions push et pop

Test dans le main: push(1, 1) (P, 2) (1, 3)

push(LIST, c)

$c = \text{pop}(\text{LIST})$

3 pop → 3, 2, 1

13 h 50

Main

TD20210607.c

List Tableau

Pile

list.c
list.h

lifo.c
lifo.h

(lifo: Last In First Out)

```
1 // lifo.h
2
3 #pragma once
4
5 #include "list.h"
6 #include "error.h"
7
8 eErrorCode push(pList l, Element e);
9 Element pop(pList l);
```

```
#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;
}
```

→ empiler un élément sur la pile

insertion de l'élément en fin de liste.

→ dépiler un élément de la pile

→ Lit, le dernier élément → stocké dans e
→ suppression du dernier.

→ renvoi de l'élément lu sur la pile.

Liste : Queue d'éléments (FIFO)

FIFO : First In First Out



put : ajout d'un élément dans la queue

get : Lecture de l'élément de tête.
Suppression de cet élément.

Utilisation de la liste-tableau

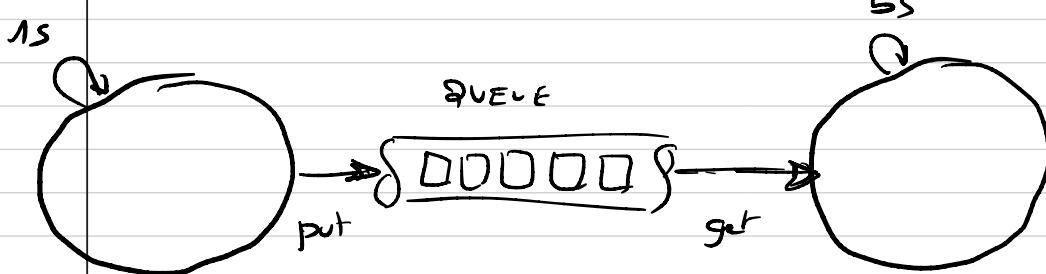
[] [] []

[1 2 3]

get ← put insertion en fin de liste

· lire le n^e élément

· suppression du n^e élément



TD 2021 06 07 :

put (list , e)
e = get (list)

main put (list , 1 2 3 puis 3)
 get 1 ; 2 ; 3 .

14h30

```
// fifo.h

#pragma once

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

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

fifo.h

fifo.c

```
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];
        | deleteElemAt(l, 0);
    }
    else {
        | printf("Error bad list.\n");
    }
    return e;
}
```

→ insertion dans la queue.

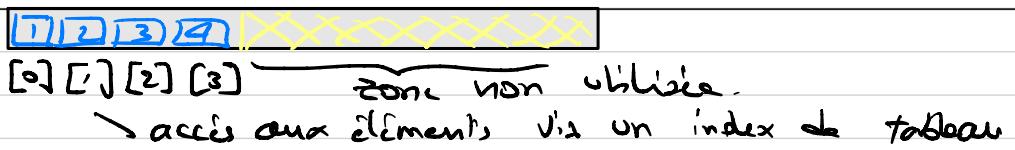
on insère un élément à la fin

→ lecture

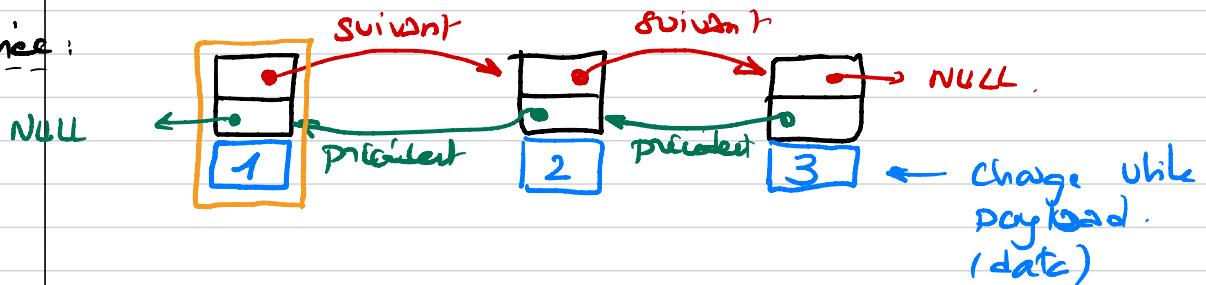
→ récupération du 1^{er} élément
+ suppression du 1^{er} —

Liste chainées -

liste tableau:



liste chaînée:



l'élément se compose de :

- 1 **pointeur** vers l'élément suivant
- 1 **pointeur** vers l'élément précédent
- 1 **charge utile** (type élément de la liste tableau)

Structur

typedef struct sElem {

 struct sElem * next;

 struct sElem * prev;

 Element e;

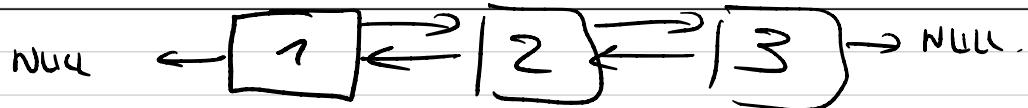
notation spéciale qui permet à la structure de contenir des éléments du même type

} sElem;

typedef struct sElem {

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

} sElem;



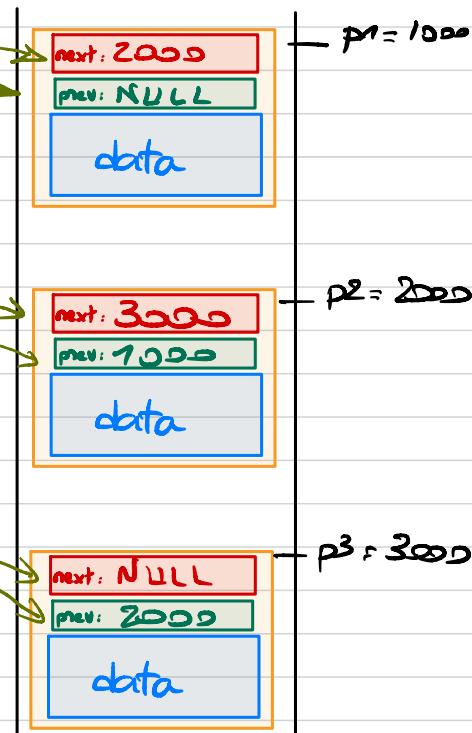
sElem * p1 = (sElem *) malloc (sizeof (sElem));
sElem * p2 = (sElem *) malloc (sizeof (sElem));
sElem * p3 = (sElem *) malloc (sizeof (sElem));

// chaining

p1 → prev = NULL;
p1 → next = p2;

p2 → prev = p1;
p2 → next = p3;

p3 → prev = p2;
p3 → next = NULL;



// list:

sElem * first = p1;
sElem * last = p3;

/* Cir. to los elementos de la lista.

sElem * e = first;

p1
p2
p3

first
last

while (e != NULL) {
 printf ("%p", e->e);
 e = e->next;

1000
2000
3000