

listes chainées

7. XI. 2021

```
typedef struct sElem {
```

```
    struct sElem *next; } Element;
```

```
} sElem;
```

2 pointeurs qui lient éléments
"avant" et "après".

données ou charge utile

TD 20210607:

- 1) Création de la structure. **sElem**
- 2) créer 3 blocs sElem pointé par p¹, p² et p³
- 3) faire le lien entre p¹ \geq p² \geq p³
- 4) les charges utiles:

p¹: 1.
p²: 2.
p³: 3.

- 5) Ajouter 2 pointeurs first qui point sur p¹ last "

[

TD 20210607.c
error.h / .h
chainedList.c / .h

IS 15

```

// chainedList.h
#pragma once
typedef struct sElem;
struct sElem {
    double e;
    struct sElem *prev;
    struct sElem *next;
};

Element e; // charge ult
} sElem;

```

```

#include "chainedList.h"

int main(int argc, char const *argv[])
{
    sElem *p1 = (sElem *)malloc(sizeof(sElem));
    sElem *p2 = (sElem *)malloc(sizeof(sElem));
    sElem *p3 = (sElem *)malloc(sizeof(sElem));

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

    p1->prev = NULL;
    p1->next = p2;
    p1->e = 1.0; ←
    p2->prev = p1;
    p2->next = p3;
    p2->e = 2.0; ←
    p3->prev = p2;
    p3->next = NULL;
    p3->e = 3.0; ←

    return 0;
}

```

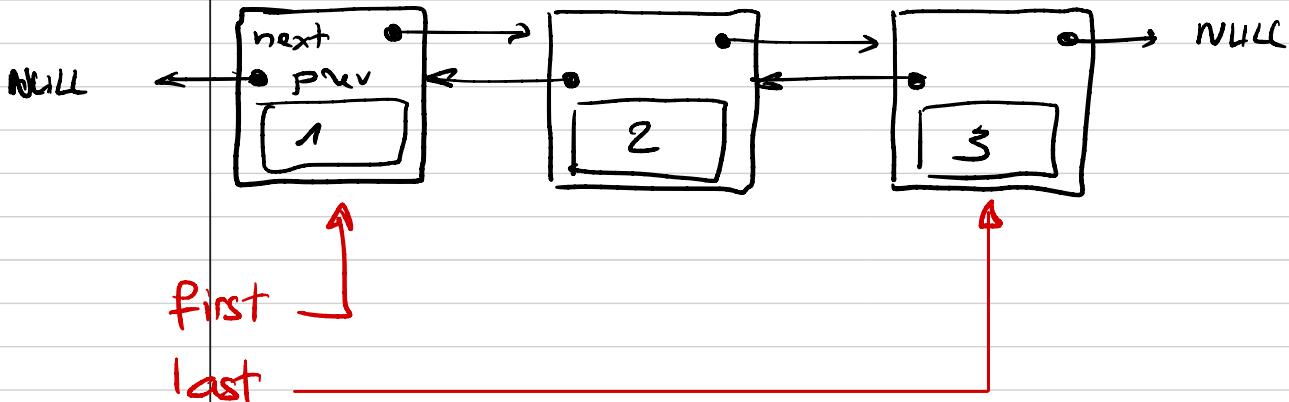
sElem est une structure récursive
Elle contient un champ qui est
un pointeur sur son propre type

) création des 3 éléments

) création de 2 pointeurs

↳ première ↳ dernier) élément

) chaînage "manuel"



1^{er} élément : first

2^{ème} élément : first → next

3^{ème} élément : first → next → next (= last)

dernier : last

avant dernier : last → prev

avant avant dernier = last → prev → prev
(= first)

Représenter et implémenter l'affichage de tous les éléments de la liste
en commençant par le premier.

```

sElem *elem = NULL;
    ^ premier
elem = first;
while(elem!=NULL) {
    printf("%lf\n", elem->e);
    elem = elem->next;
}

```

← élément "mobile" pouvant se déplacer dans la liste

← on vérifie que l'on utilise un élément qui existe
→ accès à la charge utile

on passe au suivant via le champ "next" de manière itérative

affichage :

- 1.
- 2.
- 3.

```

elem = last; ← dernier
while(elem!=NULL) {
    printf("%lf\n", elem->e);
    elem = elem->prev;
}

```

on passe au précédent

affichage :

- 3.
- 2.
- 1.

sElem : éléments de base de la liste

Description pour la liste :

: first
: last

: nombre d'éléments dans la liste

structs

typedef struct {
sElem *first;
sElem *last;
uint32_t numElem;
}
sChainedList;

implémenter cette structure et de l'utiliser pour le parcours des éléments via précédemment.

15^h 53

```

typedef struct {
    sElem *first;
    sElem *last;
    uint32_t numElem;
} sChainedList, *pChainedList;

```

) 2 pointeurs sur le 1^{er} et
dernier élément.
— nbre d'élément.

```

sChainedList l; ← nouvelle liste
l.first = p1; ) init de la liste.
l.last = p3; )
l.numElem = 3; )

sElem *elem = NULL;
elem = l.first; ↑
while(elem!=NULL) {
    printf("%lf\n", elem->e);
    elem = elem->next; suivant.
}

```

) Parcours des éléments.

Fonctions nécessaires

initialisation.

création d'une liste
first et last ← NULL
numElem ← 0

isListEmpty
getNumElem

displayList

createElem

insertElemAt

⚠ tous les éléments sont alloués dynamiquement.

(valeur de la chaîne while)
next = prev = NULL

```

eErrorCode initList(pChainedList l);
bool isEmpty(pChainedList l);
int32_t getNumElem(pChainedList l);
void displayList(pChainedList l);
pElem createElem(Element e);

```

16^h, 15

```
eErrorCode initList(pChainedList l) {
    eErrorCode returnCode = E_NO_ERROR;

    if(l!=NULL) {
        l->first = NULL;
        l->last = NULL;
        l->numElem = 0;
    }
    else {
        returnCode = E_BAD_LIST;
    }
    return returnCode;
}
```

← vérif si liste valide

```
pElem createElement(Element e) {
    pElem elem = (pElem)malloc(sizeof(sElem));
    if(elem) {
        elem->e = e; ← copy de la chg
        elem->prev = NULL; ← utili_
        elem->next = NULL; ) élément non
    } inclus dans
    return elem; la list. chaînée.
}
```

← alloc dyn de l'élément.

renvoie NULL si problème

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

} sElem, *pElem;
```

type pointeur pour sElem.

Insertion en début d. Liste

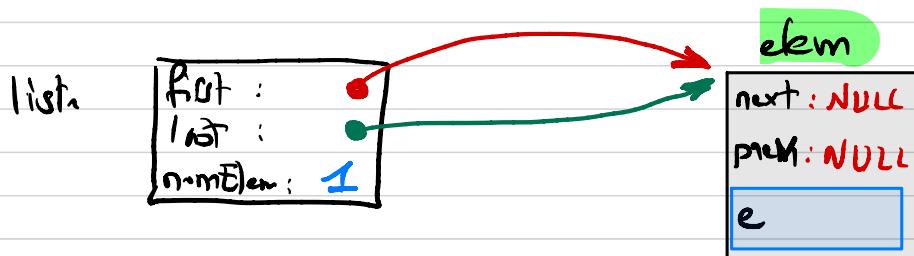
Si liste vide

Liste:

first : NULL
last : NULL
numElem : 0

1) elem = creatElem (e); // elem : c'est l'élément à insérer dans la liste

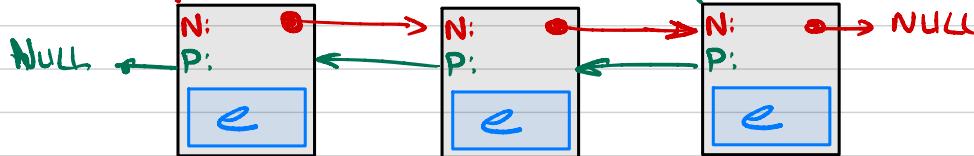
2) Placer elem dans la liste.



Si liste non vide

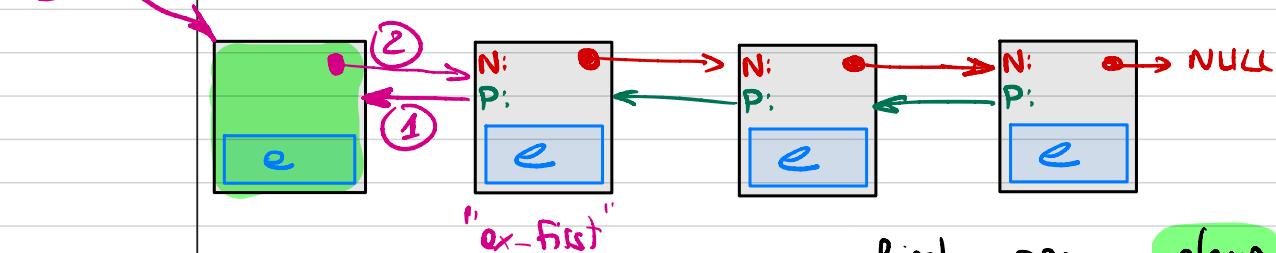
liste

first
last
numElem



1) elem = creatElem (e);

2) Places elem au début de la liste



Ordre des opérations est très important

liste → numElem++;

pour le 17. Ex. Implémenter l'insertion - || TE: tout sauf listes chainées.