# LES LISTES CHAÎNÉES

Hiver 2023

Godefroy Borduas

1

## OBJECTIF

Trouver une façon de conserver les objets selon différents critères et de les ajouter ou de les supprimer facilement.

2

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

# PLAN DE TRAVAIL

3

J'achète **4 crayons**. Comment les range-t-on ?

Construisons une boîte avec une case pour chaque crayon.

Mais si j'achète un **nouveau** cray comment puis-je le ranger ?

Construisons une nouvelle case.

Comment ajoute-t-on la nouvelle case aux autres ?

Nous pouvons les lier par une corde.

# COMMENT PEUT-ON RANGER DES CRAYONS DE COULEUR ?

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

4

Illustration de la liste chaînée

*« La liste chaînée est une **structure de donnée** qui permet de conserver une **collection ordonnée** d'objets du **même type** de taille **non définie.** »*

# LA LISTE CHAÎNÉE

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

5

- Une liste de client, trié par ordre alphabétique
- Les mots du dictionnaire
- L'inscription à un cours
- Un tableau de pointage de tous les joueurs
- L'inventaire du joueur dans un RPG.

# QUELLES SONT LES UTILITÉS DE LA LISTE CHAÎNÉE ?

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

6

Avec une liste chaînée, on peut ajouter autant d'items que l'on veut.

On peut ajouter des critères de tri ou de recherche.

On peut ajouter des conditions à l'ajout. Est-ce que le poids maximal est atteint ?

D'autres exemples ?

# COMMENT PROGRAMME-T-ON UN INVENTAIRE DE JOUEUR ?

Construire un modèle visuel | Voir d'autres **exemples** | Expliquer les inventaires dans les RPG | Programmation des listes chaînées

7

# PROGRAMMONS L'EXEMPLE DES CRAYONS

Première étape : Le crayon

Construire un modèle visuel | Voir d'autres **exemples** | Expliquer les inventaires dans les RPG | Programmation des listes chaînées

8

# QU'EST-CE QU'UN CRAYON ?

- Il est décrit par…
  - Sa couleur -> String
  - Le diamètre de sa mine (7 mm ou 5 mm) -> int
  - La force de sa mine (HB) -> String

```
#include <iostream>
#include <conio.h>

using std::cout; using std::string;

struct Crayon
{
    string Couleur;
    int Diametre;
    string Force;
};
```

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

9

# PROGRAMMONS L'EXEMPLE DES CRAYONS

Deuxième étape : La liste

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

10

- D'une « boîte » contenant le Crayon
- D'un lien vers la « boîte » suivante
- D'un indicateur de sa taille
- D'une fonction d'ajout
- D'une fonction de retrait
- D'une fonction pour récupérer
- D'une fonction pour vider la liste

**Où est-ce qu'on retrouve la première « boîte » ?**
Dans une variable *En-tête*

**Qu'est-ce qu'on entend par « boîte » ?**
Un objet qui contient quoi ?

# DE QUOI EST CONSTITUÉE UNE LISTE ?

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

11

- Le Crayon
- Le lien vers la « boîte » suivante
- Une méthode pour se rendre à la « boîte » suivante.
- Une méthode pour récupérer le crayon.

**Qui doit avoir accès à la « boîte » ?**
Seulement la liste… La « boîte » existe seulement dans la liste.

**Est-ce qu'il a des fonctions pour la « boîte » qui sont utiles ?**
Non, si la boîte n'exister que pour la liste.

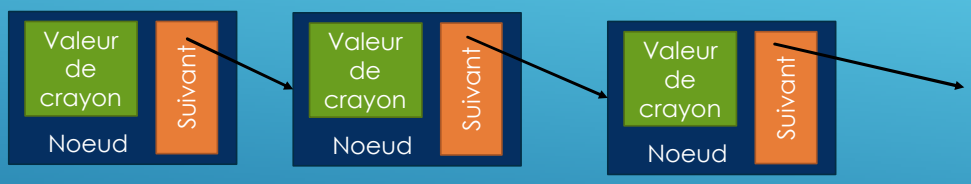
**Qu'est-ce qu'on programme en premier ?**
La « boîte », car elle est utilisée par la liste.

# QU'EST-CE QUI CONSTITUE LA FAMEUSE BOÎTE ?

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

12

```
struct Noeud
{
Crayon valeur;
Noeud* suivant;
};
```




# UN NŒUD OBJET

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

13

- Structure contenant la référence vers le premier nœud
- Contiens aussi le nombre d'éléments de la liste

```
struct Liste
{
    Noeud* racine;
    size_t taille;
};
```

# LA LISTE : L'EN-TÊTE ET LA TAILLE DE LA LISTE

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

14

- Demander la mémoire
- Établir la taille à zéro
- Établir le nœud racine à NULL
- Retourner l'adresse du début de la liste

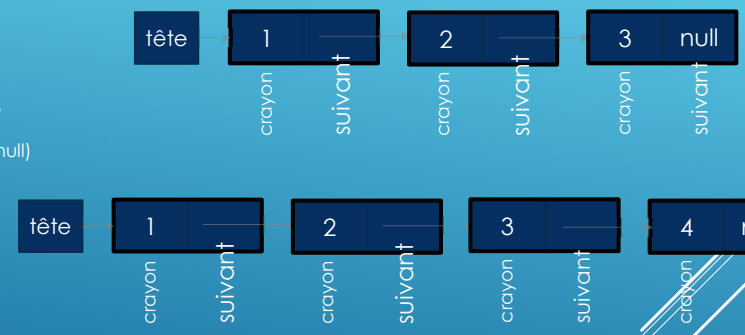
```
Liste** CreerListe() {
    Liste* liste = new Liste;
    liste->racine = NULL;
    liste->taille = 0;

    return &liste;
}

int main() {
    Liste** liste = CreerListe();
}
```

# LA LISTE : CRÉER LA LISTE

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

15

- Que doit-on faire ?
  - Créer un nœud pour le nouveau crayon
  - L'ajouter à la fin
    - Sera lié au dernier nœud
    - Quel est le dernier nœud ?
      - Celui sans lien (suivant == null)
  - Incrémenter de 1 la taille







# LA LISTE : AJOUT D'UN CRAYON (1/3)

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

16

# LA LISTE : AJOUT D'UN CRAYON (2/3)

**Complexité temporelle :**
$O(n)$

```
Liste** Ajouter(Liste** liste, Crayon c) {
    Noeud* noeud = new Noeud;
    noeud->valeur = c;
    noeud->suivant = NULL;

    // Cas où la liste est vide
    if ((*liste)->racine == NULL) {
        (*liste)->racine = noeud;
    } else {
        Noeud * node_liste = (*liste)->racine;
        while (node_liste->suivant != NULL)
        node_liste = node_liste->suivant;

        node_liste->suivant = noeud;
    }

    (*liste)->taille++;

    return liste;
}
```

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

17

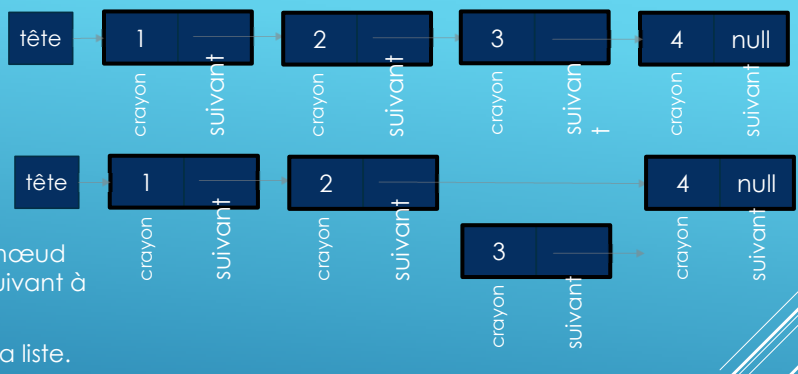
# LA LISTE : AJOUT D'UN CRAYON (3/3)

```
int main() {
    Liste** liste = CreerListe();
    Ajouter(liste, { "Noir", 7, "HB" });
}
```

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

18

# LA LISTE : RETRAIT D'UN CRAYON (1/2)




- Que doit-on faire ?
  - Changer le lien suivant du nœud précédent pour le nœud suivant à celui à supprimer
  - On se fit à la position dans la liste.
    - Plus simple à chercher (boucle for)
  - Désallouer la mémoire du noeud
  - Décrémenter de 1 la taille

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

19

```
Liste** Retirer(Liste** liste, int position) {
    if (position >= (*liste)->taille)
        return liste;

    Noeud* node = (*liste)->racine;
    for (size_t i = 0; i < position - 1; i++)
    {
        node = node->suivant;
        if (node == NULL && i < position)
            return liste;
    }

    Noeud* naretirer = node->suivant;

    node->suivant = node->suivant->suivant;
    delete naretirer;
    (*liste)->taille--;
    return liste;
}
```

# LA LISTE : RETRAIT D'UN CRAYON (2/2)

**Complexité temporelle :**
En pire cas $O(n)$

```
int main() {
    Liste** liste = CreerListe();
    Ajouter(liste, { "Noir", 7, "HB" });
    Ajouter(liste, { "Bleu", 7, "HB" });

    Retirer(liste, 1);
}
```

Construire un modèle visuel → Voir d'autres exemples → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

20

# LA LISTE : RÉCUPÉRER UN CRAYON (1/2)

- Que doit-on faire ?
  - Parcourir la liste jusqu'à la position
  - Retourner le crayon s'il existe
  - Retourner NULL sinon

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

21

# LA LISTE : RÉCUPÉRER UN CRAYON (2/2)

```
Crayon* Rechercher(Liste **liste, int position)
{
    if (position >= (*liste)->taille)
        return NULL;

    Noeud* node = (*liste)->racine;
    for (size_t i = 0; i < position; i++)
    {
        node = node->suivant;
        if (node == NULL && i < position)
        return NULL;
    }

    return &node->valeur;
}
```

```
int main() {
    /** **/
    Crayon* exemple = Rechercher(&liste, 0);
    if (exemple != NULL) {
        cout << exemple->Couleur
            << ", " << exemple->Diametre
            << " mm, " << exemple->Force
            << "\n";
        exemple->Couleur = "Bleu";
    }

    Crayon* ex2 = Rechercher(&liste, 0);
    if (ex2 != NULL)
        cout << ex2->Couleur << ", «
        << ex2->Diametre << " mm, «
        << ex2->Force << "\n";
    /** **/
}
```

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

22

# LA LISTE : LIBÉRER UNE LISTE (1/2)

- Il est toujours nécessaire de libérer la mémoire allouée
- Le programme alloue deux mémoires
  - La liste
  - Les nœuds
- Comment faire
  - Parcourir la liste (fonction récursive)
  - Libérer du dernier nœud au premier
  - Libérer la liste

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

23

# LA LISTE : LIBÉRER UNE LISTE (2/2)

```
static void LibererNoeud(Noeud* noeud) {
    if (noeud->suivant != NULL)
        LibererNoeud(noeud->suivant);

    delete noeud;
}

void Liberer(Liste** liste) {
    LibererNoeud((*liste)->racine);
    delete (*liste);
}
```

**Complexité temporelle :** $O(n)$

```
int main() {
    Liste** liste = CreerListe();
    /*
      ...
    */
    Liberer(liste);
}
```

Construire un modèle visuel → Voir d'autres **exemples** → Expliquer les inventaires dans les RPG → Programmation des listes chaînées

24

# AVANTAGE ET DÉSAVANTAGE DE LA LISTE CHAÎNÉE

25

**Avantage**
- La taille est flexible
- Possibilité de suppression d'élément
- Possibilité de réorganiser la liste (pas vu dans les diapos)

**Inconvénients**
- L'opération d'ajout est longue
- L'opération de retrait est longue
- L'opération de recherche est longue
- Récupérer un item est long

**Est-il possible d'améliorer le temps d'ajout et de retrait ?**

26