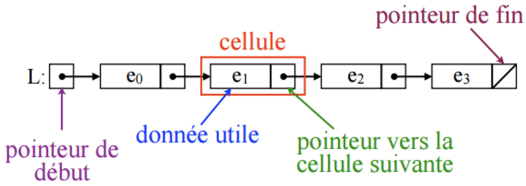
Programmation C: TP1

Ibrahim ALAME

6/12/2023

1 Liste chaînée

1.1 Définition



Une liste chaînée est composée d'un ensemble d'éléments qu'on appelle ici des cellules. On veut donc, pour ce TP, construire une liste chaînée qui contient des entiers, un entier par cellule. Et qui doit en plus avoir un pointeur dans chacune de ces cellules pour créer la liaison entre elles afin de pouvoir se déplacer tout au long de la liste chaînée. Ce pointeur doit contenir l'adresse de la cellule suivante, et si on appelle la structure utilisée `Element`, le type de ce pointeur sera alors `struct Element`.

```
typedef struct Element Element;
struct Element{
    int nombre;
    Element *suivant;
};
```

Pour une liste chaînée, il faut toujours garder dans l'esprit qu'un pointeur vers la première cellule de la liste doit être gardé quelque part, en général dans un pointeur de type struct qu'on appelle ici `Liste`.

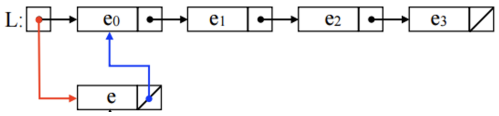
```
typedef struct Liste Liste;
struct Liste{
    Element *suivant;
};
```

1.2 Initialisation

La fonction suivante permet d'initialiser une liste chainée:

```c
Liste *initialisation(){
    Liste* liste = malloc(sizeof(*liste));
    Element *element = malloc(sizeof(*element));
    if(liste == NULL || element ==NULL) exit(EXIT_FAILURE);
    element->nombre =0;
    element->suivant = NULL;
    liste->premier = element;
    return liste;
}
```

1.3 Insertion




```c
void insertion(Liste *liste,int nvNombre){
    /* Création du nouvel élément */
    Element *nouveau = malloc(sizeof(Element));
    if(liste == NULL || nouveau == NULL) exit(EXIT_FAILURE);
    nouveau->nombre = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouveau->suivant = liste->premier;
    liste->premier = nouveau;
}
```

1.4 Supprimer un élément

```c
void suppression(Liste *liste){
    if(liste == NULL) exit(EXIT_FAILURE);
    if(liste->premier != NULL){
        Element *aSupprimer = liste->premier;
        liste->premier = liste->premier->suivant;
        free(aSupprimer);
    }
}
```

1.5 Afficher la liste chaînée

```c
void afficherListe(Liste *liste){
    if(liste == NULL) exit(EXIT_FAILURE);
    Element *actuel = liste->premier;
```

```
    while (actuel != NULL){
        printf("%d -> ",actuel->nombre);
        actuel = actuel->suivant;
    }
    printf("NULL\n");
}
```

1.6 Exemple

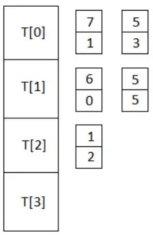
```
int main(){
    Liste* L = initialisation();
    insertion(L,4);
    insertion(L,8);
    insertion(L,15);
    afficherListe(L);
    suppression(L);
    afficherListe(L);

    return 0;
}
```

2 Représentation d'une matrice creuse

On définit une matrice creuse comme étant une matrice dont plus que la moitié des éléments sont nuls.

$$\begin{pmatrix} 0 & 7 & 0 & 5 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Pour minimiser l'espace occupé par ce type de matrice on choisi de la représenter sous forme d'un tableau de listes chaînées, de sorte que la $i^{ième}$ liste chaînée contient les éléments non nuls de la ligne i de la matrice et chacun d'eux accompagné du numéro de la colonne où il se trouve.



Tâches à faire:

1. Définir la structure qui sera utilisée pour les cellules des listes chaînées;
2. Transformer une matrice M de taille $n \times m$ en tableau de listes chaînées T comme défini précédemment;
3. Afficher un élément $M_{i,j}$ à partir de T.

3 Manipulation d'une liste chaînée

1. Écrire un programme C qui crée et lit une liste chaînée d'entiers, et lit ensuite un entier et une position et insère l'entier dans la position précisée.
2. Écrire un programme C qui crée et lit une liste chaînée d'entiers, puis supprime de cette liste toutes les occurrences d'un entier entré par l'utilisateur.
3. Écrire un programme C qui inverse une liste chaînée en manipulant seulement ses pointeurs de liaison.