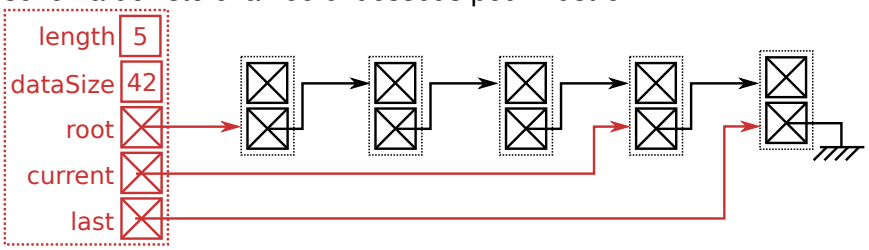
DURÉE: 1h30, UNE FEUILLE A5 RECTO-VERSO MANUSCRITE AUTORISÉE

REMARQUES:
- Les exercices sont indépendants et peuvent être réalisés dans l'ordre voulu.
- Dans l'implémentation d'une méthode, on pourra utiliser n'importe quelle autre méthode définie auparavant même si celle-ci n'a pas été implémentée.
- Dans toutes les implémentations que vous écrivez, pensez à respecter le guide de syntaxe pour la programmation (règles de nommage, présentation des blocs, *etc.*).

EXERCICE 1: QUESTION DE COURS (9 POINTS)

1. Décrire les étapes d'ajout au milieu d'un élément dans une liste. Recopier sur votre copie et compléter le schéma de liste chainée ci-dessous pour illustrer.



2. Quelle est la particularité en terme d'ajout et de suppression d'une pile comparée à une liste chaînée ?
3. Écrire une fonction permettant la création et une fonction permettant la destruction d'un volume d'entiers (tableau 3D, `int***`).
   Quelle est la compléxité de ces méthodes ?
4. On considère un système électrique (simplifié) constitué d'un fusible et d'un ensemble d'interrupteurs. Un fusible peut supporter un courant maximum (en Ampères) et pourra être fonctionnel ou cassé. Un interrupteur pourra être allumé ou éteint et aura une consommation de courant (s'il est allumé).
   Donner la déclaration des structures permettant de résoudre ce problème.

EXERCICE 2: NUAGE DE POINTS (11 POINTS)

On considère les structures suivantes permettant de manipuler un ensemble de points dans le plan.

```c
// Dans point.h
typedef struct {
  int x, y;     // Coordonees
} Point;
```

```c
// Dans nuageDePoints.h
typedef struct {
  ListeSC* points;     // Ensemble de points 2D
} NuageDePoints;
```

1. **Gestion d'un point (`point.c`)**
   (a) Donner l'implémentation de la fonction `Point createPoint( int x, int y )` qui créé et initialise un point.
   (b) Une fonction de destruction est-elle nécessaire ? Si oui, donner son implémentation.
   (c) Écrire une fonction `void printPoint( Point pt )` qui affiche un point sous la forme : `(x, y)`.
   (d) Écrire une fonction `float distance( Point pt1, Point pt2 )` qui calcule la distance entre 2 points.
2. **Liste chainée (`listeSC.c`) :** Compléter les fonctions `void freeData(void *d)` et `void afficherData(void *d)` permettant la destruction et l'affichage d'un point du nuage.
3. **Nuage de points (`nuageDePoints.c`) :**
   (a) Donner l'implémentation d'une fonction `NuageDePoints creerNuageDePoints( )` permettant de créer un nuage de points. La liste de points sera vide.
   (b) Donner l'implémentation d'une fonction `void freeNuageDePoints( NuageDePoints* nuage )` permettant de détruire un nuage de points.
   (c) Les ensembles de points sont sauvegardées dans des fichiers textes. Chaque ligne du fichier contient les coordonnées d'un point sous la forme `x y`.
   Donner l'implémentation d'une fonction `NuageDePoints readNuageDePoints( char* fName )` permettant de lire un nuage de points dans un fichier texte.
   (d) Écrire une fonction `Point getBarycentre( NuageDePoints nuage )` qui calcule le barycentre de l'ensemble de points.
4. **Fichier principal :** donner le code du programme principal. On déclarera en constante le nom du fichier contenant le nuage de points d'entrée `"file/nuage.txt"`. Après avoir lu le nuage de points, on calculera et affichera son barycentre.