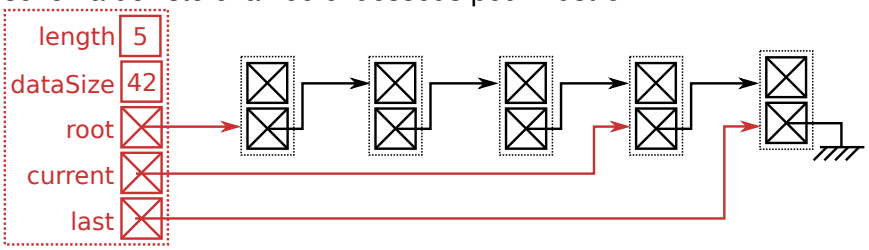
DURÉE: 1h30, UNE FEUILLE A5 RECTO-VERSO MANUSCRITE AUTORISÉE

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.*).

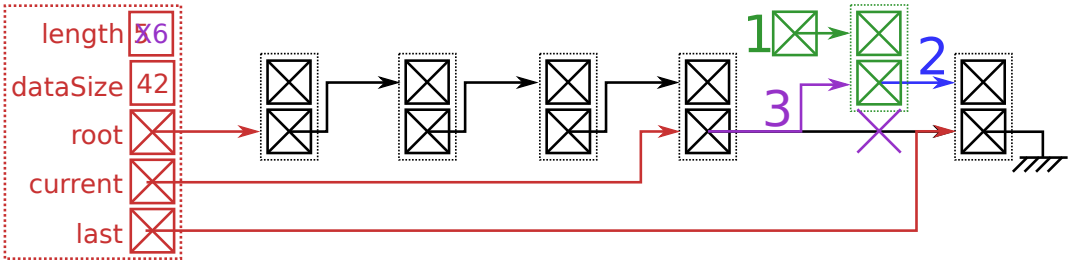
EXERCICE 1: QUESTION DE COURS (9 POINTS)

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.



Solution: (2 points) : 1 point pour les étapes, 1 point pour le schéma

1. Création d'une maille contenant le nouvel élément
2. Chainage : le champ next de la nouvel maille pointe vers le successeur de l'élément courant
3. Ajout : le champ next de l'élément courant pointe vers le nouvel élément



2. Quelle est la particularité en terme d'ajout et de suppression d'une pile comparée à une liste chaînée ?

Solution: (1 point)
Pour une pile, l'ajout et la suppression se font uniquement en tête de liste (pas d'ajout ou de suppression possibles au milieu ou à la fin).

3. Écrire une fonction permettant la création et une fonction permettant la destruction d'un volume d'entiers (tableau 3D, `int***`).
Quelle est la complexité de ces méthodes ?

Solution: (3 points) : 1 point par fonction, 1 point pour la complexité (2 boucles $\Rightarrow \mathcal{O}(n^2)$).
- Tolérer l'usage de malloc plutôt que de calloc lors de la création (les valeurs dans le volume ne sont pas précisées)
- Tolérer l'utilisation d'un `int ***` dans freeVolume. `int ****` permet de se souvenir dans la fonction appelante que `vol = NULL` (ce qui est perdu avec un `int***`).

```
int*** creerVolume( int dimX, int dimY, int dimZ ) {
  int*** vol = malloc( dimX * sizeof( int** ) );
  int i, j;
  for( i = 0; i < dimX; i++ ) {
    vol[i] = malloc( dimY * sizeof( int* ) );
    for( j = 0; j < dimY; j++ )
```

```
      vol[i][j] = calloc( dimZ, sizeof( int ) );
  }
  return vol;
}
void freeVolume( int ****vol, int dimX, int dimY ) {
  int i, j;
  for( i = 0; i < dimX; i++ ) {
    for( j = 0; j < dimY; j++ )
      free( *vol[i][j] );
    free( *vol[i] );
  }
  free( *vol );
  *vol = NULL;
}
```

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.

Solution: (3 points) : 1 point par structure
- Tolérer l'utilisation d'un tableau d'interrupteurs au lieu d'une liste (si l'étudiant a ajouté un entier pour la taille du tableau)
- Attention à l'ordre dans la déclaration des structures (Systeme après les 2 autres)

```
typedef struct {
  float courantMax;
  int fonctionnel;  // 0: casse, 1: ok
} Fusible;

typedef struct {
  float courant;
  int allume;  // 0: eteint, 1: allume
} Interrupteur;

typedef struct {
  Fusible f;
  ListeSC *interrupteurs;
} Systeme;
```

EXERCICE 2: NUAGE DE POINTS (11 POINTS)

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

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

```
// 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.

Solution: (0.5 point)

```
Point createPoint( int x, int y ) {
  Point pt = {x, y};
  return pt;
}
```

   (b) Une fonction de destruction est-elle nécessaire ? Si oui, donner son implémentation.

Solution: (0.5 point)
Comme il n'y a pas d'allocation dynamique, il n'est pas nécessaire de prévoir une fonction de destruction.

(c) Écrire une fonction `void printPoint( Point pt )` qui affiche un point sous la forme : `(x, y)`.

Solution: (0.5 point)

```
void printPoint( Point pt ) {
  printf( "(%d, %d)\n", pt.x, pt.y );
}
```

(d) Écrire une fonction `float distance( Point pt1, Point pt2 )` qui calcule la distance entre 2 points.

Solution: (0.5 point)

```
float distance( Point pt1, Point pt2 ) {
  float dist = sqrt( (pt1.x - pt2.x)*(pt1.x - pt2.x) + (pt1.y - pt2.y)*(pt1.
    ↪ y - pt2.y) );
  return dist;
}
```

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.

Solution: (1 point)

```
void freeData( void *d ) {
  // Rien a faire (pas d'allocation dynamique)
}

void afficherData( void *d ) {
  Point *pt = (Point*)d;
  printPoint( *pt );
}
```

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.

Solution: (1 point)

```
NuageDePoints creerNuageDePoints( ) {
  NuageDePoints nuage;  // Creation de la structure
  nuage.points = creerListe( sizeof( Point ) );
  return nuage;
}
```

(b) Donner l'implémentation d'une fonction `void freeNuageDePoints( NuageDePoints* nuage )` permettant de détruire un nuage de points.

Solution: (1 point)

```
void freeNuageDePoints( NuageDePoints* nuage ) {
  freeListe( nuage->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.

Solution: (2 points) : 1 point pour la bonne gestion des fichiers (*i.e.* ouverture, test et fermeture)

```
NuageDePoints readNuageDePoints( char* fName ) {
  NuageDePoints nuage;
  // Ouverture en lecture d'un fichier texte
```

```
  FILE *fTxt = fopen( fName, "r" );
  if( fTxt != NULL ) {  // On s'assure que le fichier a bien ete ouvert
    nuage = creerNuageDePoints( );  // Creation du nuage
    // Lecture et ajout de points
    int x, y;    // Coordonnees
    while( fscanf( fTxt, "%d %d\n", &x, &y ) == 2 ) {
      Point pt = createPoint(x, y);    // Nouvel element
      ajout( nuage.points, &pt, 2 );   // Ajout a la fin
    }
    fclose( fTxt );    // Fermeture du fichier
  }
  return nuage;
}
```

(d) Écrire une fonction `Point getBarycentre( NuageDePoints nuage )` qui calcule le barycentre de l'ensemble de points.

Solution: (2 points) : 1 point pour la bonne déclaration de la boucle for, 1 point pour le calcul (avec le cast vers un Point*)
Attention : les coordonnées d'un point doivent être entières.

```
Point getBarycentre( NuageDePoints nuage ) {
  int x = 0, y = 0;  // Coordonnees du barycentre
  // Somme des coordonnees en x et y des points de la liste
  for( nuage.points->current = nuage.points->root; hasNext( nuage.points );
     ↪ getNext( nuage.points ) ) {
    Point *pt = (Point*)(nuage.points->current->data);
    x += pt->x;
    y += pt->y;
  }
  // Division entiere comme x, y, et length sont entiers
  int n = nuage.points->length;
  Point barycentre = creerPoint( x / n, y / n );
  return barycentre;
```

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.

Solution: (2 points) 0,5 point pour l'inclusion des bibliothèques, 0,5 point pour la définition de constante (`#define`), 0,5 point pour la destruction du nuage à la fin du programme.

```
#include "point.h"   // Pas obligatoire (incluse dans nuageDePoints.h)
#include "nuageDePoints.h"

#define FILE_NUAGE "file/nuage.txt"

int main( ) {
  NuageDePoints nuage = readNuageDePoints( FILE_NUAGE );
  Point barycentre = getBarycentre( nuage );
  printPoint( barycentre );
  freeNuageDePoints( *nuage );

  return 0;
}
```