Ce sujet a été crée dans le cadre d'une séance de Tutorat en Algo C/C++. Il se base sur les sujets d'années précédentes de DS d'Info 4 Spé et d'Info 5 Spé. Ce sujet ne sera possiblement pas un reflet exact du DS ou examen des matières correspondantes. Les questions sont prévus pour une durée d'une heure.

1 Timeline

Contexte Le principe va être de créer une frise chronologique simplifiée en utilisant des structures, de la récursivité et des listes chaînées. Voici le code donné pour la structure `Timeline` :

```
struct Timeline {
    Event* head;
};
```

Une `Timeline` sera symbolisé par une liste chaînée d'événements organisée par ordre croissant. Nous aurons aussi une structure `Event` qui permettra de décrire un événement via une année `year`, un libelle `label`, ainsi qu'un pointeur vers le prochain élément.

Vous devrez donc initialiser cette frise chronologique, ajouter des éléments à cette frise chronologique durant l'initialisation et après-coup, ainsi que l'afficher. Nous ne demanderons pas ici de fonctions pour supprimer des éléments.

En annexe vous sera donné un exemple de la fonction `main`.

Questions

1. Écrivez la structure `Event` avec les attributs décrits précédemment.

   Solution:

   ```
   struct Event {
       int year;
       string label;
       Event* next;
   };
   ```

2. (a) Écrivez la fonction `ajouterTimeline`, une *fonction récursive*, avec le prototype suivant :

   ```
   void ajouterTimeline( Timeline* tl, int remaining );
   ```

   Cette fonction devra :
   - Demander les informations à l'utilisateur pour remplir une variable de type `Event`.
   - Chercher le bon endroit où placer cet `Event` dans la `Timeline`.
   - Rappeler la fonction en décrémentant le nombre d'événements restants à ajouter.

   Conseil : Réfléchissez d'abord au cas d'arrêt de votre fonction. Pensez à initialiser aussi vos pointeurs à `NULL` pour éviter les fuites mémoires.

   (b) Écrivez une fonction `initTimeline` qui prendra en paramètre un pointeur vers un objet `Timeline`. Cette fonction demandera à l'utilisateur le nombre d'événements qu'il veut ajouter via la fonction `ajouterTimeline`.

Solution:

```
void ajouterTimeline( Timeline* tl, int remaining ){

    if( remaining == 0 )
        return;

    cout << "Nombre d'evenements restants a ajouter : " << remaining <<
        endl;

    // Initialisation de l'Ã©vÃ©nement.
    Event* ev = new Event;
    cout << "En quelle annee s'est passe votre evenement ? ";
    cin >> ev->year;
    cout << "Quel est le libelle de votre evenement ? ";
    cin >> ev->label;
    ev->next = nullptr;

    // RÃ©cursive.
    if( tl->head == nullptr ){
        tl->head = ev;
    } else {
        Event* current = tl->head;
        Event* previous = tl->head;

        while( current != nullptr && ev->year > current->year ){
            previous = current;
            current = current->next;
        }

        ev->next = current;
        previous->next = ev;
    }

    // Rappel de la fonction.
    ajouterTimeline( tl, remaining - 1 );
}

void initTimeline(Timeline* tl){
    int remaining;
    cout << "Combien d'Ã©lÃ©ments voulez-vous ajouter dans la timeline ?
        ";
    cin >> remaining;
    ajouterTimeline( tl, remaining );
}
```

3. Écrivez une fonction `ajouterEvenement` qui appellera la fonction `ajouterTimeline` avec un

nombre d'événements à ajouter de 1.

Solution:

```
void ajouterEvenement(Timeline* tl){
    ajouterTimeline( tl, 1 );
}
```

4. Ajoutez une fonction `afficherTimeline` qui va afficher de bas en haut le contenu de la timeline comme dans le rendu suivant :

```
/*
Exemple avec une Timeline de deux elements : {2025,"Debut de votre
    annee";2026,"Fin de votre annee"}

---
 |
 # : 2025 - Debut de votre annee universitaire
 |
 # : 2026 - Fin de votre annee universitaire
 |
\_/
*/
```

Votre `Timeline` ne devra pas être forcément passé par pointeur. Si la `Timeline` est vide, elle affichera le message « Vide » plutôt que d'afficher la frise.

Solution:

```
void afficherTimeline( Timeline tl ){

    if( tl.head == nullptr ){
        cout << "Vide" << endl;
    } else {
        cout << "___" << endl;

        Event* ev = tl.head;
        while( ev != nullptr ){
            cout << " | " << endl;
            cout << " # : " << ev->year << " - " << ev->label << endl;
            ev = ev->next;
        }
        cout << " | " << endl << "\\_/";
    }
```

2 File d'attente au cinéma

Contexte Vous voulez aller voir le dernier blockbuster sorti au cinéma. Mais une file de n personnes attendent aussi d'entrer dans la salle pour assister à la séance. Vous devez donc vous mettre en queue de la file, et vous vous mettez à calculer dans combien de temps vous allez pouvoir rentrer dans la salle. Sachant qu'il faut une minute à une personne pour entrer dans la salle et être placé par un employé.

Vous devrez donc créer une structure `File` qui comportera des éléments de la structure `Personne`. Puis créer une fonction `main` qui enfilera des personnes (10 par exemple) et qui affichera le temps que vous devrez attendre avant d'entrer dans la salle.

Questions

1. Ajoutez une structure `File`, avec un seul attribut qui sera la queue de la file de type `Personne`, ainsi que ses principales fonctions.
   (a) `initFile`, avec un pointeur de `File` passé en paramètre, et qui va mettre la queue de la file à la valeur `NULL`.
   (b) `enfilerPersonne` qui va enfiler une Personne déjà crée avant. Elle prendra donc en paramètre votre file, ainsi qu'un objet `Personne`.
   (c) `defilerPersonne` qui va enlevera la personne à la tête de la file et retournera son nom.
   (d) `fileVide`, une fonction qui renverra un booléen pour savoir si la file, passé en paramètre est vide.

Conseils : Pensez à vider la mémoire quand vous faites une opération de défilage. Voici un exemple de la structure `Personne` :

```
struct Personne {
    string nom;
    Personne* prochain;
};
```

Solution:

```
struct Personne {
    string nom;
    Personne* prochain;
};

struct File {
    Personne* queue;
};

void initFile( File* f ){
    f->queue = nullptr;
};
void enfilerPersonne( File* f, Personne* p ){
    if( f->queue == nullptr ){
        f->queue = p;
```

```
    } else {
        Personne* pr = f->queue;
        p->prochain = pr;
        f->queue = p;
    }
}
string defilerPersonne( File* f ){

    Personne* current = f->queue;
    Personne* previous = nullptr;

    while( current->prochain != nullptr ){
        previous = current;
        current = current->prochain;
    }

    string nom = current->nom;

    if(previous == nullptr){
        f->queue = nullptr;
    } else {
        previous->prochain = nullptr;
    }

    delete current;
    return nom;
}
bool fileVide( File f ){
    return f.queue == nullptr;
}
```

2. Écrivez la fonction `main` qui devra :
   (a) Créer une variable de type `File` et l'initialiser.
   (b) Ajouter dix personnes dans la file. Le nom donné aux personnes n'a pas d'importance et peut-être identique entre toutes les personnes.
   (c) Compter le temps à attendre pour vous afin d'entrer dans la salle.

   Vous devrez utiliser *toutes les fonctions* définies dans la question précédente.

Solution:

```
int main(){
    File f;
    initFile( &f );

    for( int i = 0; i < 10; i++ ){
        Personne* p = new Personne;
```

```
        p->nom = "a";
        p->prochain = nullptr;
        enfilerPersonne( &f, p );
    }

    int temps_attente = 0;

    while( !fileVide( f ) ){
        defilerPersonne( &f );
        temps_attente++;
    }

    cout << "Je dois encore attendre " << temps_attente << " minutes."
        << endl;
    return 0;
}
```

3 Tris et complexité

1. Écrivez l'algorithme du tri par sélection, tel que vue dans le cours. Indiquez également la complexité de chaque ligne.
2. Sans écrire l'algorithme, veuillez expliquer le principe du tri fusion et nous donner sa complexité dans le meilleur des cas, ainsi que dans le pire des cas.

Solution: Voir cours Tris.

A Fonction main pour la Timeline

```
int main(){
    Timeline tl;
    tl.head = nullptr;

    initTimeline( &tl ); // Saisie des Ã©lÃ©ments.
    ajouterEvenement( &tl );
    afficherTimeline( tl );

    return 0;
}
```