

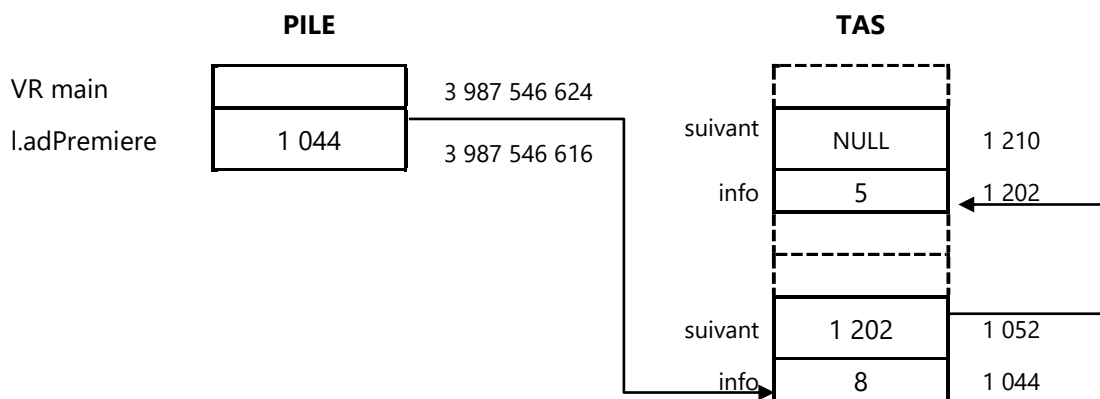
TD10 : Liste chaînée

On supposera dans tout ce TD que les listes sont simplement chaînées non circulaires.

Exercice 1 : Dynamique des données en mémoire

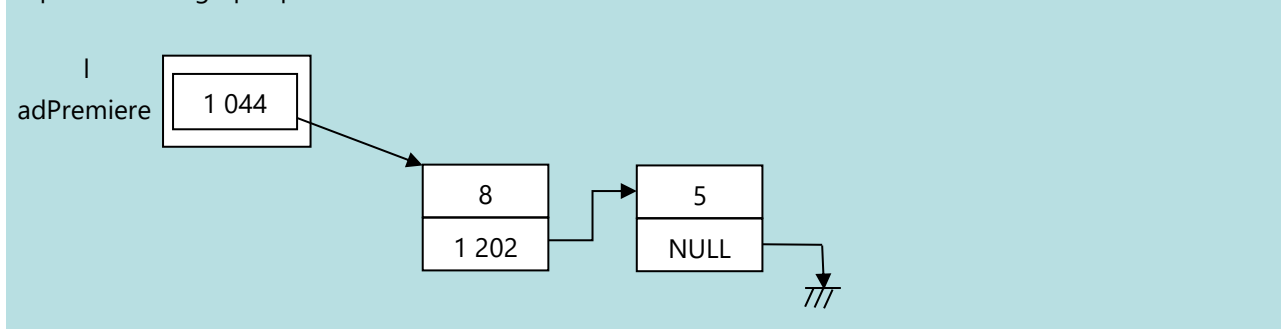
Considérons le programme principal incomplet et l'état de la mémoire suivants :

```
int main () {
    Liste l;
    l.ajouterEn.....(8);
    l.ajouterEn.....(5); /* Etat de la mémoire dessiné à ce moment */
    ..... /* Instruction à rajouter*/
    ..... /* Instruction à rajouter*/
    return 0;
}
```



- a. Utilisez la représentation graphique présentée en cours pour décrire la liste créée par ce programme.

Représentation graphique :



- b. Complétez les pointillés des appels `l.ajouterEn.....(8)` et `l.ajouterEn.....(5)`.

```
l.ajouterEnQueue(8) // ou Tete
l.ajouterEnQueue(5)
```

- c. Rajoutez dans le programme principal les deux instructions qui permettent :

- d'afficher l'information contenue dans la première cellule.
- d'afficher l'information contenue dans la deuxième cellule.

```
cout << (l.adPremiere)->info << endl;
cout << (l.adPremiere)->suivant->info << endl;
```

Exercice 2 : Création d'une liste à partir d'un tableau

Écrivez un constructeur de la classe Liste qui, à partir d'un tableau dynamique d'éléments, crée la liste contenant les mêmes éléments dans le même ordre. Donnez un exemple d'appel à ce constructeur. On supposera que ElementTD et ElementL sont des types compatibles.

Dans le fichier Liste.h :

```
Liste (const TableauDynamique & tab);
```

Dans le fichier Liste.cpp :

```
Liste::Liste (const TableauDynamique & tab) {
    Cellule * cel;
    adPremiere = NULL;
    for (int i=tab.taille_utilisee-1; i>=0; i--) {
        cel = new Cellule;
        cel->info = tab.valeurIemeElement(i); // affectation OK car types compatibles
        cel->suivant = adPremiere; // suivant pointe sur l'ancien élément de tête
        adPremiere = cel; // mise à jour de l'élément de tête
    }
}
```

Remarque : insérer un élément en tête de liste coûte 1 new, 3 affectations de pointeurs, et 1 affectation d'élément. Ainsi, le coût d'un ajout en tête est $O(1)$. Donc le coût total de la création de liste est $O(n)$.

Exemple de programme principal, dans le cas où les types ElementTD et ElementL correspondent au type double :

```
#include "TableauDynamique.h"
#include "Liste.h"

int main () {
    TableauDynamique tab;
    tab.ajouterElement(1.0);
    ... // autres ajouts
    Liste l (tab);
    l.afficher();
    return 0;
}
```

Exercice 3 : Inversion d'une liste

Écrire en C++ une procédure globale qui inverse l'ordre des éléments d'une liste chaînée passée en paramètre, sans faire de delete ni de new.

```
void inverserListe (Liste & l) {
    Cellule * adPremiereListeInverse = NULL;
    Cellule * maCellule;

    while (l.adPremiere != NULL) {
        // on prend la première de l et on la met en tête de la liste inversée
        maCellule = l.adPremiere;
        l.adPremiere = l.adPremiere->suivant;
        maCellule->suivant = adPremiereListeInverse;
        adPremiereListeInverse = maCellule;
    }
}
```

```
l.adPremiere = adPremiereListeInverse;// la liste inversée commence à adPremiereListeInverse
}
```

Exercice 4 : Suppression des occurrences d'un élément

Ecrire une procédure membre qui supprime toutes les occurrences d'un élément e dans une liste.

L'élément e peut apparaître plusieurs fois de suite et aussi en tête de liste. Pour ne pas faire trop de cas particuliers, on ne va traiter qu'à la fin le cas où e apparaît en tête.

```
void Liste::supprimerElements (ElementL e) {
    Cellule * p = adPremiere;
    Cellule * pred;
    if (p == NULL) return ; // liste vide

    /* On examine la liste en commençant par la deuxième cellule */
    pred = p;
    p = p->suivant;
    while (p != NULL) {
        if (estEgalElementL(p->info, e)) { // on a trouvé une occurrence
            pred->suivant = p->suivant;
            delete p;
            p = pred->suivant;
        }
        else { // on continue à chercher
            pred = p;
            p = p->suivant;
        }
    }
    /* A ce moment, toutes les occurrences de e ont été supprimées,
    sauf peut-être la première si jamais la cellule de tête contenait e */
    if (estEgalElementL(adPremiere->info, e)) {
        p = adPremiere;
        adPremiere = p->suivant;
        delete p;
    }
}
```