Algoritmen I LABO: Inleidende oefening op pointers en gelinkte lijsten

Hieronder staat de definitie van een lijst.

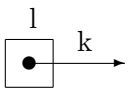
```
template <typename T>
class Lijstknoop;

template <typename T>
class Lijst{
    public:
        //...
    private:
        Lijstknoop<T>* k;
};
```

Laten we deze afspraak maken voor onze tekeningen:

- een Lijst is een vierkant
- een Knoop is een vierkant met afgeronde hoeken
- de locatie waar een pointervariabele opgeslagen wordt, wordt aangegeven met een zwart bolletje - terwijl de pijl die vandaaruit vertrekt aanduidt waarnaar de pointer wijst
- de naam van een pointer zetten we soms op het zwarte bolletje, soms ook op de pijl (omdat daar doorgaans meer plaats is)
- een sleutel (zie verder) is een cirkel

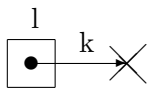
Je ziet dat een lijst slechts 1 datalid heeft, nl. een pointer `k` naar een knoop. Schrijven we in het main-gedeelte van ons programma de declaratie `Lijst l;`, dan wordt dit zo weergegeven op tekening:



Merk op dat de naam `l` 'absoluut' is, terwijl de naam `k` 'relatief' is. Hiermee bedoelen we het volgende:

1. In de code van de main kan je de lijst `l` rechtstreeks aanspreken met `l`.
2. In de code van de main kan je de pointer `k` niet rechtstreeks aanspreken met `k`, maar zal je `l.k` moeten schrijven. (O ja, dit zal wel compileerfouten geven omdat `k` in de klasse `Lijst` een private datalid is, maar laten we daar even abstractie van maken...)

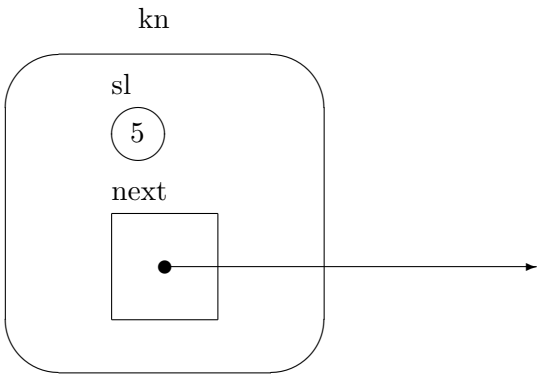
Nog op te merken: we lieten de pointer `k` op de tekening voorlopig nog niet expliciet naar een knoop wijzen - er werd nog geen geheugen gealloceerd bij deze pointer. We zeggen dan dat `k` een zwevende pointer is - en die zijn TE MIJDEN!! Je weet echter uit de cursus C/C++ dat een nette constructor van de klasse `Lijst` de pointer `k` op 0 zal initialiseren. Dus wordt onze tekening eigenlijk zoals hieronder.



In beide schetsen is het *niet* af te leiden uit de tekening van welk type de pointer `k` is. Dat moet je nog zelf ergens (in je achterhoofd) noteren. Of, bij de zwevende pointer, een Knoop in stippellijnen tekenen om aan te geven dat er een Knoop zou *kunnen* aanhangen (maar dat nog niet doet).

Hieronder de definitie en declaratie van een Knoop, gevolgd door de bijhorende schets.

```
template <typename T>
class Lijstknoop{
   friend class Lijst<T>;
   public:
      Lijstknoop(const T &d = T());
   private:
      T sl;          // sleutel
      Lijst<T> next;
};
```



Hier geldt weer: `kn` is een absolute naam, `sl` en `next` zijn relatief ten opzichte van `kn`.

In het vervolg van de oefening zal jou gevraagd worden om de schetsen te maken, en de bijhorende namen op tekening / in code met elkaar in verband te brengen.

1. Stel dat de lijst `mijn_lijst` drie knopen bevat, met sleutels 3, 6 en 9. Teken deze lijst, en geef hier dan de volledige naam van de laatste pointer (i.e. de nullpointer). Deze naam begint met `mijn_lijst ...`. Vanaf nu zullen we ervoor zorgen dat alle knopen in de lijst geordend staan volgens stijgende sleutelwaarde.

2. Als we aan de lijst een sleutel (en dus knoop) willen toevoegen, moeten we eerst zoeken waar hij moet komen. Teken in vorige schets een nieuwe pointer met naam `h` die naar de tweede knoop wijst. Merk op: deze pointer is géén onderdeel van de gelinkte lijst. We kunnen de laatste pointer uit onze tekening (de nullpointer) nu ook benoemen startend vanuit `h`. Doe dit. (Niet dat we die nullpointer voor iets nodig hebben, maar louter als oefening op het aflezen van gegevens uit een tekening.)
   De plaats waar pointer `h` nu staat, maakt het mogelijk om tussen de knoop met sleutel 6 en die met sleutel 9 een knoop tussen toe voegen. We doen dit in de volgende stap. Let op: vanaf nu spreek je elke pointer/Lijst/Knoop uit de tekening aan met een naam die begint met `h` (en dus niet met `mijn_lijst`).

3. Herneem de tekening drie, vier of vijf keer. In elke tekening zet je één stap, zodat er uiteindelijk een nieuw knoop met sleutel 7 tussen de tweede en de derde knoop hangt. Zet telkens de juiste code (= 1 regel) onder de tekening.

4. Stel dat we nu een knoop willen toevoegen met sleutel 12. Naar welke knoop moet de hulppointer `h` dan wijzen, om de code van vorige stap te kunnen hergebruiken? En naar welke knoop moet `h` wijzen voor het invoegen van een knoop met sleutel 4?

5. Indien we een knoop met sleutel 2 willen toevoegen aan de lijst, waar moet `h` dan naar wijzen?

6. Je ziet dat dit niet kan: er ís geen knoop waar je `h` naar kan laten wijzen. Nu zijn er 2 mogelijkheden:

   (a) je voorziet een `if/else`-structuur om het speciale geval op te vangen (= extra werk en duplicated code)

   (b) je herwerkt het algemene geval zó dat ook dit ene geval meteen opgelost raakt (= extra werk)

   Uiteraard kiezen we voor de tweede methode, omdat we te wijs zijn om ons in te laten met duplicated code.

7. Dus: herbegin met de lijst met drie knopen, zoek naar het object uit die lijst waar je de pointer `h` naar laat wijzen, en ga na of toevoegen van sleutel 7 én sleutel 2 met dezelfde code kan. Schrijf deze code op dezelfde manier als hierboven: bij elke verandering aan de tekening, schrijf je 1 regel code.
   Tip: stel dat je een knoop met sleutel 7 wil toevoegen. Als je de pointer `h` laat wijzen naar de knoop met sleutel 9 staat `h` te ver naar rechts. Laat je de pointer `h` wijzen naar de knoop met sleutel 6, dan staat hij (blijkens ons probleem geconstateerd in vorige opdracht) teveel naar links. Mik ergens tussen beide posities in!

8. Onthoud hier vooral het TYPE van de pointer `h` uit. Dit zal je nodig hebben bij de eerste vraag van labo 1, nl: wat is het returntype van de functie `zoek_gerangschikt`.