Uppgift 2

Implementera en operator som jämför en sträng med ett heltal. Operatorn ske ge ut `true` om strängen har färre tecken än heltalet har siffror, annars `false`.

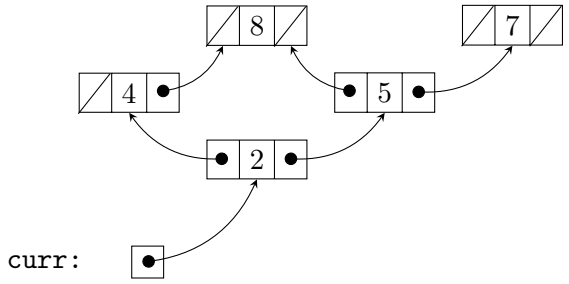
KRAV: Din operator ska implementeras som en fri funktion.

TIPS: Förväntad lösning är ca 4 rader kod.

TIPS: Konvertera heltalet till en sträng och räkna tecken.

Uppgift 3

Skriv kod som skapar den länkade strukturen nedan.



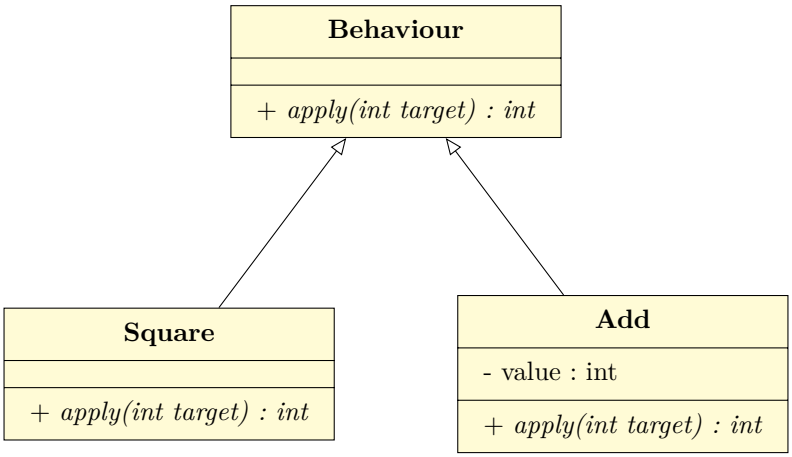
KRAV:

- Du måste använda den givna `Node` från `uppg3.cc`.
- Samtliga element ska vara dynamiskt allokerade. Du behöver inte avallokera elementen.
- Du får inte deklarera några fler variabler än de som redan är deklarerade i `main()`.

TIPS: Förväntad lösning är 2 - 7 rader kod.

Uppgift 4

Implementera följande klasshierarki:



- `Behaviour::apply(int target)` har inget definierat beteende.
- `Square::apply(int target)` returnerar `target * target`
- `Add::apply(int target)` returnerar summan av `target` och datamedlemmen `value`

KRAV:

- Alla dina klasser ska vara deklarerade och definierade i *en* källkodsfil.
- Den givna filen `uppg4.cc` har ett main-program som ska fungera oförändrat.

Funktionella krav: Körning av programmet ska ge följande utdata i terminalen:

```
100
```

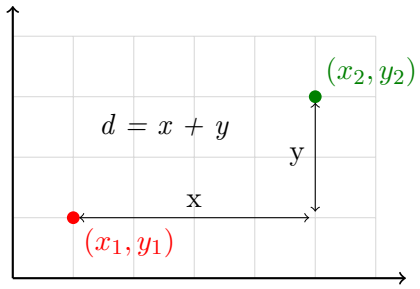
TIPS: Förväntad lösning är ca 34 rader kod.

Del 2 - Uppgift 5

Ett lagerhus förvarar varor som ska till ett antal butiker. En lastbil utgår från lagerhuset och kör ut till alla butiker. För att hantera butiksdestinationerna och hitta den optimala vägen behöver lastbilen en klass som håller koll på var lagerhuset är och vart alla destinationer är.

Din uppgift är att implementera två datatyper enligt instruktionerna nedan.

Destination är ett aggregat (`struct`) som sparar ett namn på en plats samt två heltal x och y för att representerar platsens koordinater. Utöver det ska du implementera en fri funktion för att beräkna avståndet mellan två destinationer. Lastbilen kan inte köra fågelvägen (då skulle den krascha in i hus) så avståndet beräknas som "manhattanavståndet". D.v.s. att vi summerar distansen i x-led och distansen i y-led. Observera att distansen alltid ska vara positiv.



Route_Planner är en klass som ger en lastbilschafför information om de destinationer hen ska besöka. Klassen behöver lagra

- en lista med destinationer. Listan representerar de butiker som ska besökas.
- en destination som representerar lagerhuset.

Med ett objekt av klassen ska man kunna

- beräkna den totala distansen en lastbil skulle behöva köra om den åkte fram och tillbaka från lagerhuset till varje destionation i listan.
- lägga till en till destination i listan genom att anropa `operator+=`. Om den ny totala distansen enligt ovan skulle bli större än 100 ska destinationen inte läggas till i listan. Istället ska ett undantag kastas enligt körexemplet.
- plocka ut den destination i listan som är närmast en given startdestination.
- skriva ut alla destinationer i listan på given utström genom `operator<<`.

Körexempel

```
Med 'ica' blir den totala distansen större än vad en lastbil klarar av!
=== Alla butiker ===
# Total distans: 94
willys: [10, 10]
coop: [3, -1]
hemköp: [-8, 4]
lidl: [-4, -5]

======
Närmaste butik är: lidl
```