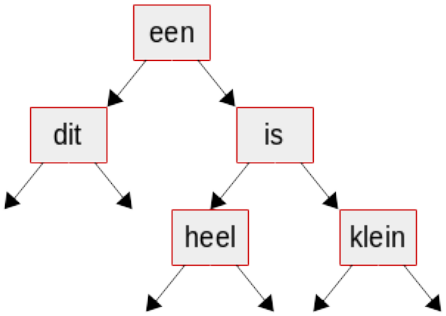
BINAIRY TREE

In deze opgaven gaan we kennis maken met een veelgebruikte datastructuur: **de binairy tree**.
Hierbij heeft elke structuur **twee** pointers naar een andere instantie van die structuur. In de code noemen we deze '*links*' en '*rechts*'.
Een structuur (woord met pointers links en rechts) wordt hier een '**node**' genoemd.



Binairy tree's worden o.a. toegepast in indexen van database tabellen. Onder bepaalde voorwaarden (balancing) kan een btree een optimale zoek mogelijkheid bieden.

Voor de 'organisatie' van de binairy tree in deze opgave geldt:

- elke 'node' heeft een woord.
- Alle woorden die 'kleiner' zijn worden ondergebracht in de *linker* tak.
- Alle woorden die 'groter' zijn worden ondergebracht in de *rechter* tak.
- Dit geldt voor elk niveau.

In deze opgaven gaan we op 2 verschillende manieren bepalen wat groter/kleiner is.

1. String compare : volgens de alfabetische ordening v/d woorden.
2. Int compare : volgens de integer volgorde van de *lengte van de woorden*.

Door een keuze te maken voor ordening 1 of 2, krijgen we dus een andere opbouw van de binairy tree.

Codeer instructies :

Zie '**compare**' paragraaf verderop.

1. Maak de 2 compare functies **strCompare** en **strLengteCompare**
   zie het commentaar in de header file.
2. Maak de functie addToBtree(…).
   hierbij wordt een node gemaakt met het woord, en op de juiste positie opgenomen in de binairy tree.
   Bij elke node, te beginnen bij de node waar **rootptr** naar wijst, wordt mbv de opgegeven compare functie bepaald of de nieuwe node links (compare geeft <=0)of rechts compare geeft >0) moet worden toegevoegd.
   Uiteindelijk vinden we zo de pointer die leeg is (waarde NULL) , deze moet dan gaan verwijzen naar de nieuwe node.
   Zie ook : *functie-pointer.pdf*
3. Maak tenslotte de exportBtree functie. Deze zal alle woorden uit de binairy tree opnemen in een stringstream. (Deze wordt al reference meegegeven aan de functie.)
   De volgorde is : eerste wat links staat, dan het woord van de node, dan wat rechts staat.
   Na elk woord moet een spatie komen.
   Deze functie '*schreeuwt*' om een recursieve oplossing!
   Zie ook : *recursie.pdf*

BINAIRY TREE

Gegeven is de volgende code :

btree.cpp:

```
#include <iostream>
#include <sstream>
#include <string>
#include "btree.h"

… hier alle functies uit de header file …

int main()
{  // voorbeeld invulling !
    std::string lijst[] = {"Denkend", "aan", "Holland" };

        int aantal = 3;
        Node* strRoot = NULL, *intRoot = NULL;
        for (size_t i = 0; i < aantal; i++) {
                addToBtree(&strRoot, lijst[i], &strCompare);
                addToBtree(&intRoot, lijst[i], &strLengteCompare);
        }
        std::stringstream ss;
        exportBTree(strRoot, ss);
        std::cout << ss.str() << std::endl;
         // evenzo export en display voor intRoot !
    return 0;
}
```

Merk op:

1. De inhoud van de '***main***' routine is niet van belang voor de test.
   Je mag hier test-code opnemen voor de gemaakte functies, geheel naar eigen inzicht.
2. De 'signatuur' van de functies mag niet worden aangepast. Wijzigingen geven fouten in de test.

Compare

De compare functie vergelijkt twee variabelen (eerste en tweede) en returned een int :

- 0 als beide gelijk zijn.
- getal > 0 als eerste > tweede
- getal < 0 als eerste < tweede

```
int intCompare(int eerste, int tweede){
        if ( eerste == tweede)
                return 0;
        else
        if (eerste > tweede)
                return 1;
        else
                return -1;
}
```

BINAIRY TREE

in functie vorm : int compare(<type> eerste, <type> tweede). [waarbij type van alles kan zijn]
Bij sommige classen (b.v. string) is compare gedefinieerd als een methode v/d klasse:
eerste.compare(tweede) waarbij eerste en tweede beide string zijn.

Upload instructies:

Upload de volgende bestanden:

1. btree.cpp