Middle East Technical University Department of Computer Engineering

CENG 213

Data Structures

Fall '2015-2016

Programming Assignment 2

Due date: 13 December 2015, Sunday, 23:55

1 Objectives

This assignment aims to make you familiar with tree structures and basic operations on trees. You are asked to build tree structures from parent-child relations and you will implement depth first and breadth first traversals on those trees.

Keywords: *Tree, Traversal, Search*

2 A Simple Wordnet

WordNet[1] is a lexical database for the English language. It groups English words into sets of synonyms called synsets. All synsets are connected to other synsets by means of semantic relations. There are several different relations between synsets. In WordNet, nouns are organized into hierarchies, defined by hypernym or IS A relationships. Opposite of hypernymy is hyponymy. A hyponym shares a type-of relationship with its hypernym. For example, pigeon, crow, eagle and seagull are all hyponyms of bird (their hypernym); which, in turn, is a hyponym of animal. In this assignment you will only work with these relations and build small WordNet's with IS A relations. We will call these relations superclass and subclass relations in this assignment and use them to perform several operations on these simple WordNet's.

[1]https://wordnet.princeton.edu/

3 Operations

The WordNet class must contain at least the following two functions:

3.1 BuildWordnet

The first interface method is the BuildWordNet function that takes a filename containing the relations as its argument and builds a tree for the WordNet.

```
void WordNet::BuildWordNet(string filename){
  // read the contents of the input file
  // build a tree for wordnet
}
```

The input file will contain the root as its first element. After the root each node in the tree is presented with *is_a* relations. The format will be as *child is_a parent*. Introduction of a node will occur after the introduction of its parent but the order of relations will not follow any other strategy. You must insert the siblings in the given order. There will be no errors or missing information in the input file.

```
standart
scale is_a standart
richter_scale is_a scale
medium_of_exchange is_a standart
currency is_a medium_of_exchange
coin is_a currency
money is_a medium_of_exchange
fund is_a money
nickel is_a coin
dime is_a coin
```

The WordNet will be as follows:

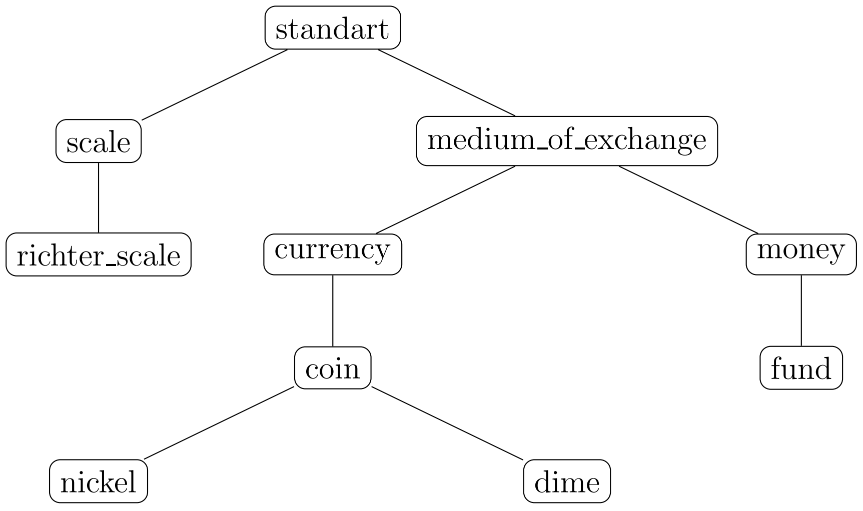

Figure 1: WordNet build from the example relations file

3.2 HandleTask

The second method is the entry point for many tasks that you will perform on the WordNet. It will again take a string as its argument. The string format is presented for each task in the following subsections. Format and output examples are given in the next section.

```
void WordNet::HandleTask(string command){
  // parse the task string and perform the desired task
}
```

3.2.1 PrintWordNet

The first task is to print the WordNet. The task command is simply *PrintWordNet*. The output will be in depth first manner. Each line will contain a word and each level will be indented by two spaces.

3.2.2 PrintSubclasses

PrintSubclasses task prints the subclasses (node, its children and all descendants) of a given class in the breadth first order. Each node will be printed to a new line. Command format is *PrintSubclasses word*.

3.2.3 PrintSuperclasses

PrintSuperclasses task prints the superclasses (node, its parent and all ancestors) of a given node. Output will be in single line. Nodes will be separated with " < " (space, less than, space) characters. Command format is *PrintSuperclasses word*.

3.2.4 PrintIntermediateClasses

PrintIntermediateClasses task prints the superclasses (node, its parent and all ancestors) of a given node up to the second node. Output will be in single line. Nodes will be separated with " < " (space, less than, space) characters. Command format is *PrintIntermediateClasses word1 word2*.

4 Examples

In this section, example task commands and related outputs are presented for each task.

4.1 PrintWordnet

```
>>> PrintWordNet
standart
  scale
    richter_scale
  medium_of_exchange
    currency
      coin
        nickel
        dime
    money
      fund
```

4.2 PrintSubclasses

```
>>> PrintSubclasses standart
standart
scale
medium_of_exchange
richter_scale
currency
money
coin
fund
nickel
dime
```

```
>>> PrintSubclasses currency
currency
coin
nickel
dime
```

```
>>> PrintSubclasses fund
fund
```

4.3 PrintSuperclasses

```
>>> PrintSuperclasses standart
standart
```

```
>>> PrintSuperclasses fund
fund < money < medium_of_exchange < standart
```

```
>>> PrintSuperclasses currency
currency < medium_of_exchange < standart
```

4.4 PrintIntermediateClasses

```
>>> PrintIntermediateTypes medium_of_exchange medium_of_exchange
medium_of_exchange
```

```
>>> PrintIntermediateTypes fund medium_of_exchange
fund < money < medium_of_exchange
```

```
>>> PrintIntermediateTypes coin medium_of_exchange
coin < currency < medium_of_exchange
```

5 Regulations

1. **Programming Language:** You will use C++.
2. **E**xternal libraries are not allowed.
3. **Late Submission:** Refer to the syllabus for late submission policy.
4. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations.
5. **R**emember that students of this course are bounded to code of honor and its violation is subject to severe punishment.
6. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

6 Submission

- Submission will be done via COW.
- Create and submit ***hw2.tar.gz*** that contains ***tree.hpp, tree.cpp, wordnet.hpp*** and ***wordnet.cpp*** without any folders.
- Do not write a *main* function in any of your source files.
- You will only implement *tree* and *wordnet* classes.
- Do not remove or modify any variable or function given in the classes. You can add any variables or functions to *tree* and *wordnet* classes.
- Compile and test your source codes as follows:

```
g++ -ansi -pedantic -Wall main.cpp tree.cpp wordnet.cpp -o hw2
./hw2
```

- A test environment will be ready in Moodle.
  - You can submit your source files to Moodle and test your work with a subset of evaluation inputs and outputs.
  - This will not be the actual grade you get for this assignment.

7 References

This section is a reference material for several useful classes and functions from the standard library.

7.1 String

```
#include <string>

string str1;
string str2;
str1.compare(str2); # returns 0 when strings are equal
```

7.2 Vector

```
#include <vector>

vector<Type> myvector; # replace type with int, char etc. or a class name
myvector.push_back(x); # adds a new element at the end of the vector
myvector[i];           # returns the element at position i in the vector
myvector.size();       # returns the number of elements in the vector
```

7.3 Stack

```
#include <stack>

stack<Type> mystack; # replace type with int, char etc. or a class name
mystack.push(x);     # inserts a new element at the top of the stack
mystack.empty();     # returns whether the stack is empty
mystack.pop();       # removes the element on top of the stack
mystack.top();       # returns the top element in the stack
mystack.size();      # returns the number of elements in the stack
```

7.4 Queue

```
#include <queue>

queue<Type> myqueue; # replace type with int, char etc. or a class name
myqueue.push(x);     # inserts a new element at the end of the queue
myqueue.empty();     # returns whether the queue is empty
myqueue.pop();       # removes the next element in the queue
myqueue.front();     # returns the next element in the queue
myqueue.size();      # returns the number of elements in the queue
```