# Calculating "The Sum of Its Parts"

## Prerequisites, Goals, and Outcomes

**Prerequisites:** Students should have mastered the following prerequisite skills.

- Trees - Knowledge of tree representation, particularly general hierarchies
- STL map container - Knowledge of use of STL map container, including iteration
- Recursion - Knowledge of creating a recursive solution for a problem

**Goals:** This assignment is designed to reinforce your understanding of trees and use of a tree based STL container.

**Outcomes:** Students successfully completing this assignment would master the following outcomes.

- Understand tree representation
- Use of the STL map container
- Produce a recursive algorithm to traverse a tree representation

## Background

"The whole is greater than the sum of its parts" is a central theme of Gestalt psychology, and for humans it may be a true observation. Computers are much more literal-minded.

## Description

In this assessment, you will construct a representation of a complex structure, a hospital that is composed of many parts. There are too many parts for a human to keep track of, which is why we want the computer to do it for us. The description of the hospital is contained in the file definitions.txt .
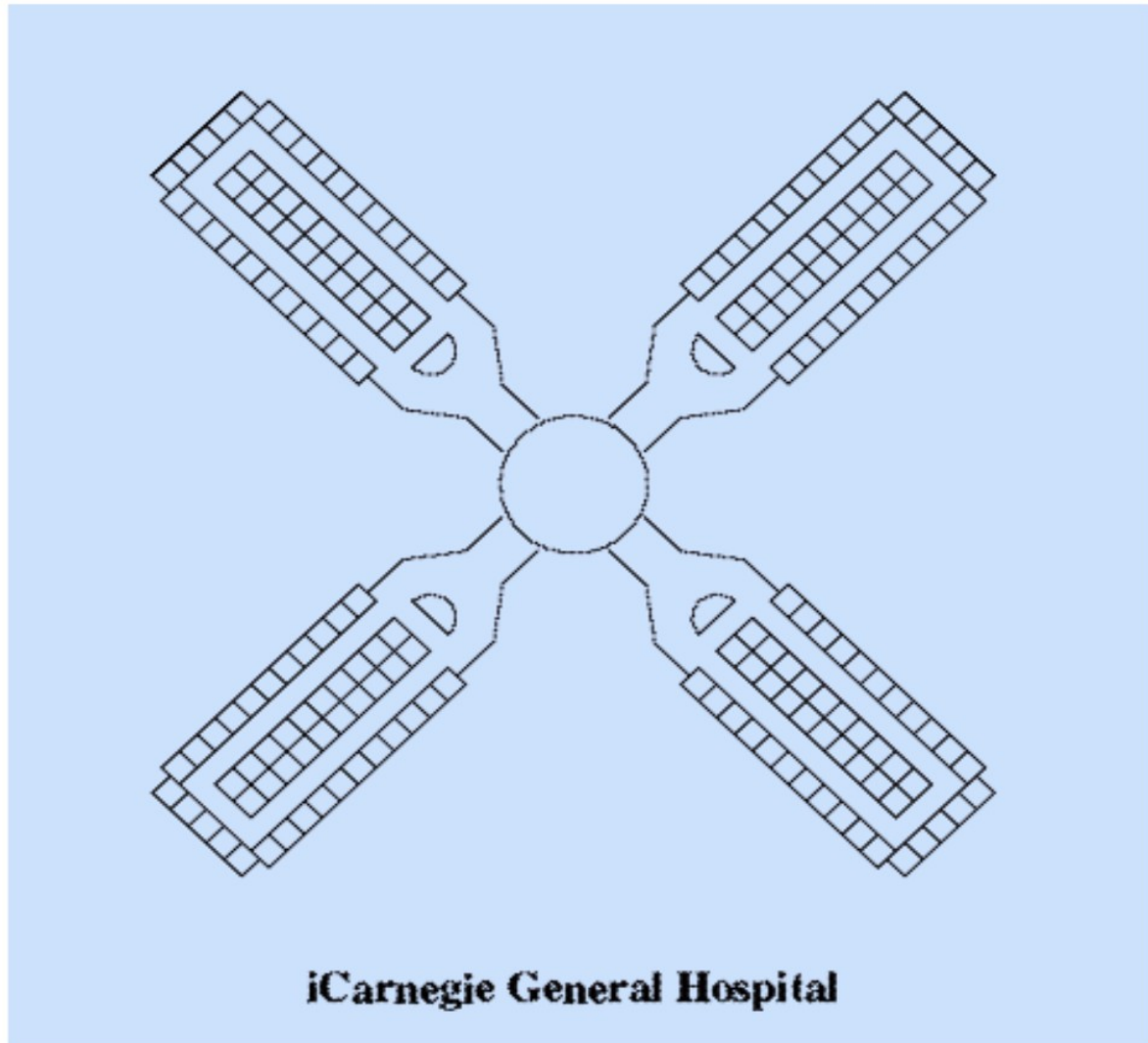


Figure 1 Layout of a floor in the hospital

The building has ten floors. Each floor has four wings emanating from a central core.

Each wing contains two long corridors joined at the end by a short connecting corridor.

Each long corridor contains twenty-one patient rooms. Each connecting corridor contains five supply rooms.

The hospital is described by a labeled tree, whose nodes are of type Part . A node contains children nodes corresponding to its subparts, as shown in the figure below. Each edge is labeled by the number of subparts the node contains. For example, the label 10 on the edge from hospital to floor indicates that the hospital has ten floors. You can assume there are no duplicate edges, that is, there is at most one edge between any two nodes.
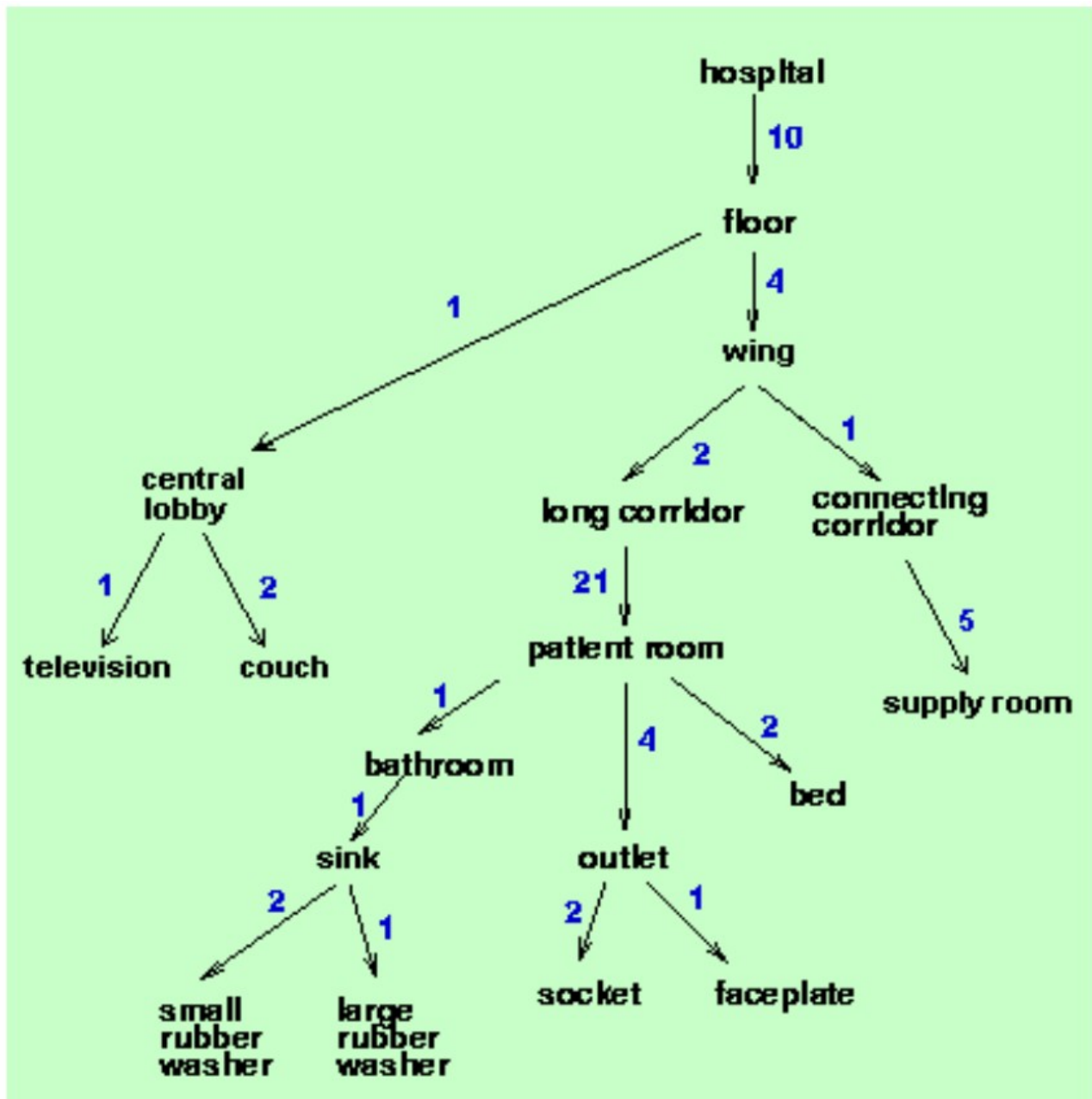
Figure 2 Tree representation of the hospital

The driver program *main.cpp* first loads the file *definitions.txt* , which contains the subpart relationships that define the hospital. It then processes queries from the file queries.txt , which contains two kinds of queries about the hospital. The whatis query requests the description of a Part . The howmany query is the heart of the exercise. It asks how many instances of a Part are contained in another Part .

You are provided with main.cpp and a skeleton version of parts.h as a starting point. Your job is to complete the parts.h file (and write *parts.cpp* , if you deem necessary). The steps below point the way toward a solution, but they do not cover every detail, so if you find you need to create additional functions or member items, feel free to do so.

## Files

Following is a list of files needed to complete this assessment.

contains all of the following necessary files:
- *main.cpp* - Testing program needed for you to check your solution
- *parts.h* - Skeleton version of the header file
- *definitions.txt* - Definition of part and subpart relationships
- *queries.txt* - File containing queries to test your implementation.

## Tasks

To complete this assessment, you need to complete the implementation of class Part and other helper functions.
To begin, verify the files needed for this assessment.

1. **Extract** the archive to retrieve the files needed to complete this assessment.

Following is an ordered list of steps that serves as a guide to completing this assessment. Work and test incrementally. Save often.

2. **Begin** by adding to class Part an additional member named subparts that is a container of type map . Your container should map a pointer (of type Part ) to an integer. So, for example, the Part named hospital will have among its subparts an entry mapping a pointer to the Part named floor to the int 10, since a hospital contains 10 floors.

3. **Next**, implement the describe method of class Part . It should list the part name and all of the subparts and their quantities. Use iterators to process the subparts. Here are some examples of how output of describe should look:

> Part long_corridor subparts are:
> 21 patient_room
>
> Part floor subparts are:
> 4 wing
> 1 central_lobby
>
> Part couch subparts are:
> It has no subparts.

4. **Then**, implement the recursive method *countHowMany(Part const \*p)* of class Part that counts the number of instances of the part pointed to by p that occur in the invoking object. Counting proceeds by simple recursion on the subparts. For example, asking hospital how many wing subparts it contains, we see a hospital has 10 floors. Recursively asking floor how many wing subparts it contains, we see that a floor has four wings. Calling the recursive function again, we ask wing how many times it occurs in wing. The function returns the value 1 since this is the base case. The final answer is then 40 (10 \* 4 \* 1).

5. **Next**, complete the implementation of method *lookup* of class NameContainer , which maps a string to a pointer to an instance of type Part . Method lookup should automatically create an instance of class Part with the given name if one does not already exist in the map. The method should return a pointer to the instance.
Finally, implement method *addPart*(string const &part, int quantity, string const &subpart)

that adds the part named y as a new subpart of the Part named x .
This method is called by *loadDefinitions* in *main.cpp* . To obtain pointers to the
Part objects specified by the string parameters, use method l*ookup* of object
partContainer .

## Submission
Submit **only** the following.

1. parts.h - your completed version, not the skeleton
2. parts.cpp - if created