Lab Assignment 3.

Lab Mon June 28. Due Wed Jun 30, 23:00.

Data Structures Engi4892 2004. T. Norvell

0 In-place, linked list, recursive quicksort.

Design, implement, and test a set of subroutines that will rearrange the nodes in a null-terminated, singly-linked list to be in sorted order using the recursive quicksort method. Note that no nodes should be created or destroyed by your subroutines.

The signature for your subroutine will be

```
void quickSort( Node *&head ) ;
```

where Node is defined by

```
struct Node { int data; Node *next ; } ;
```

Notes:

- Websubmit as file quicksort.cpp, assignment 3, part 0.
- Bonus Marks: Of the submissions that pass all tests and have an "acceptable level of style marks", the fastest implementation on a random list will be given a 50% bonus. Please carefully document any design decisions that you make in the name of speed.

1 Search Engine

Design, implement, and test a query method for search engines.

Internet search engines must be able to identify which documents contain which words. The more useful search engines allow complex queries using boolean operators such as 'and', 'or' and 'not'. For this assignment you write a subroutine that calculates which documents match a given query.

For querying purposes, each document can be regarded as a set of words. Our entire database of documents can be regarded as a function from document name (URL) to the set of words in the document. E.g. the test database is this function

$$
\begin{array}{lll}
 & database & \\
= & \{\ (www.fwords.com, & \{foo, firkin, farthing, funk, ...\}), \\
 & (www.music.com, & \{funk, rock, jazz, punk, ...\}), \\
 & (www.unkious.com, & \{funk, sunk, punk, ...\})\ \}
\end{array}
$$

To make querying more efficient, this database has been processed to yield an index. The index is a function from each word to the set of all names of documents containing the word

$$
\begin{array}{lll}
 & index & \\
= & \{\ (funk, & \{www.fwords.com, www.music.com, www.unkious.com\}), \\
 & (punk, & \{www.music.com, www.unkious.com\}), \\
 & (jazz, & \{www.music.com\}), \\
 & (sunk, & \{www.unkious.com\}), \\
 & ... & \}
\end{array}
$$

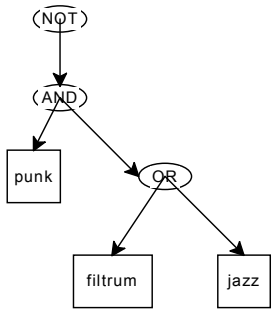
Also precomputed is the set of all filenames:

$$all = \{www.fwords.com, www.music.com, www.unkious.com\}$$

Queries, are typed in by the user as strings such as:

NOT (punk AND (filtrum OR jazz))

These have been stored in the computer memory as trees such as:[0]



[0] The problem of getting from a string to a tree representing its structure is called "parsing" and is quite important also in designing compilers and other language processors. See Chapter 5 (section on defining languages) of the text for some ideas on how to parse strings using recursive routines.

I have represented these trees using 4 classes. There is a class for AND nodes, a class for OR nodes, a class for NOT nodes, and a class for STRING nodes (i.e. the nodes that represent single word queries).

All 4 of these classes are derived from (i.e. inherit from) a class called QueryNode.[1] The inheritance structure looks like this:

```
class QueryNode { ... } ;
class StringNode : public QueryNode {... } ;
class AndNode : public QueryNode { ... } ;
class OrNode : public QueryNode { ... } ;
class NotNode : public QueryNode { ... } ;
```

Instances of the last three classes are connected by pointers to the nodes below them. For example, the AndNode object in the illustration has two pointers; one (its leftChild_) points to the StringNode object for "punk"; the other (its rightChild_) points to the OrNode object:

```
class AndNode : public QueryNode {
    ...
    private:
        QueryNode *leftChild_ ;
        QueryNode *rightChild_ ;
} ;
```

Instances of the StringNode contain their word as a ch_string.

Class QueryNode declares a virtual function evaluate but does not define it.[2]

[1] QueryNode is an example of a "abstract class'. I.e. it is a class we do not intend to create direct instances of; its purpose is to serve as a base class for other classes.

[2] Recall that with virtual functions the function definition used depends on the actual class of recipient object. Suppose p is declared

```
QueryNode *p ;
```

but it contains the address of an AndNode (which is perfectly legitimate, since AndNode is derived from QueryNode). A call

```
p->evaluate(...)
```

will call the function AndNode::evaluate,rather than QueryNode::evaluate.

```
class QueryNode
{
    public :
        // Evaluate the query for a given database.
        virtual StringSet evaluate( const Index &index, const StringSet
        &all) = 0 ;
...
} ;
```

The purpose of evaluate is to calculate the set of document names matching the query. The type it returns —StringSet— is simply an abbreviation for Set<ch_string>, and the type of its first parameter —Index— is simply an abbreviation for Function<ch_string, StringSet>.

Design, implement, and test the evaluate subroutine for each of the 4 classes derived from QueryNode.

Put all 4 function definitions (together with any auxiliary functions) in a file called `query_tree.cpp` and submit it as assignment 3.

Notes:

- Your file should #include file `query_tree.h`. This should include anything else you need.
- Your 4 functions will be mutually recursive: e.g. to evaluate an AND node, first evaluate its left and right parts and then combine the results appropriately.
- I have defined a second virtual function member called print (see file query_print.cpp). It may give you some inspiration.
- If a word is not in the domain of the index, then it does not occur in any document in the database. Be careful not to apply the index function to something not in its domain.
- All files needed to test your definitions will be made available at www.engr.mun.ca/~theo/Courses/ds/. Let me know immediately if you have any problems with these files.
- Put all your code in a file query_tree.cpp. Websubmit as assignment 3 part 1.