Grosvenor: A New Cognitive Theory With a Computational Implementation

April 2003

Roeland Hancock
rhancock@presencedesigns.net

Team 70, Silver High School

Abstract

One the greatest challenges of man has been to understand himself. A multitude of theories model the mind, but these are only theories—no operating implementation has ever been constructed. Computational technology has now progressed to the point that a partial, if not full, implementation of a mental model can be constructed. One of the most important aspects of a cognitive model is its ability to learn. A successful creation and implementation of such a model will provide an extremely revealing look at the mind and revolutionize cognitive science.

Introduction

This project's goal is the development and implementation of a new cognitive model. The model will be a cross-disciplinary approach drawing from many sources with most of the foundation based on the “building-block” approach (Minsky 1988).

Background and Theory

This project's ultimate goal is a cognitive model and creation of the first true artificial intelligence (AI), but will have many side applications of immediate practical and commercial use, primarily as a search engine, expert system and reference tool. These commercial opportunities will no doubt be explored in the course of future development as a means of financing further work. This goal will be resource and time intensive; Grosvenor is not expected to come to fruition in the available time frame.

Although this project does present a new, cross disciplinary theory of cognition, it does not stop at mere theory, but goes on to suggest and even implement said theory. Thus, the theory is not so much concerned with an unquestionable and assuredly valid theory, but a broad theory which takes in the most important elements of cognition, defines a plausible framework and assembles a working theory which can, in a computational environment, be implemented, providing the first functional machine intelligence, allowing for testing and refinement of the theoretical foundations, and unprecedented analysis of the mind. This computational environment will not be a model of a mind, it will be a mind.

One of the most fascinating problems of the mind is child learning, specifically linguistic learning. Children are born with little or no innate knowledge, yet they master speech, mathematics, commonsense knowledge and eventually ask "why?". Older children still learn but are of lesser interest as they may be taught new ideas by building on what they have already learned—a capability not of particular interest in bottom-up AI. Developing an AI and allowing it to truly develop from the bottom up is akin to a child's sensorimotor and preoperational stages of development (Piaget ; Kolb 1984). Somehow children acquire, sometime between birth and age 7, the fundamental knowledge and understanding of the world that will enable them to learn or be taught essentially anything else. In the language of computers, the child would be "bootstrapped", stored memory and instructions dumped into an accessible, usable area. But a child is not injected with knowledge, it accumulates it through interaction and observation of the world.

Consider how a child learns a word. The school age child may be handed a vocabulary sheet with new words, defined in presumably known words, perhaps the word is used in a

sample sentence. This is teaching and is a trivial matter. But how does a child learn the prerequisite words? How can something be explained when the underlying concepts must also be explained, and the concepts underlying those must be explained, etc? No one defined a child's first word so that the child used the word appropriately. The child heard a sequence of noises being used frequently in its environment, gradually became aware that it could reproduce such sounds itself, and finally reproduced the sounds, in sequence, triggering a positive response from the environment, most likely parents. At this point the word is not even a thought of as a coherent concept but as discrete vocal elements. Psychology would treat this explanation as a stimulus-response pair and leave it at that. S-R is only a surface view, however, and cannot be reproduced. The psychological view is flawed in that actual functional mechanisms have not yet been explored on a level of reproducibility and nothing has actually been explained.

The study of of the mind copes with language is interesting not only with regard to development, but how close the ties between cognition and language are (Chomsky 1965 & 1968, Chambers 2002, Saffral 2002).

One recurring issue in the theories discussed here is motivation. Piaget demonstrated that infant will pursue a goal for their own delight and, in the process, learn and develop (Piaget 1966). This project will assume likewise and avoid becoming bogged down with a philosophical basis for its actions; simply learning is fulfillment.

The child has a framework of associative memories which is empty at birth. Immediately, the child begins to observe and store information, not as individual, unconnected pieces of information, but as knowledge sets, a loosely related collection of information. The depth of each piece of information is unknown and rather arbitrary.

The sentence on a vocabulary sheet is a reinforcement of contextual information and helps to tie together the defining words.

Consider the word "gudajsdie". These letters do not actually form a "real" word and are treated as nonsense. Now consider the sentence "A gudajsdie IS A ertsid, kjhsdfu bmsel FOR sadaING." The sentence has several more nonsense words, but also "a", "is" and "for", known words which signify certain relationships between words in a sentence. By adding such contextual cues, you can assume that "gudajsdie" is a singular, indefinite noun which can be described using the words "ertsid", "kjhsdfu" and "bmsel" and is involved in the action of "sadaing". If you were asked about "gudajsdie", you would be able provide information about the part of speech and its connection with the other nonsense words, *despite having no clear knowledge of what the word* means. If you are exposed to several more sentences containing the same nonsense words and contextual cues, you may begin to construct a network and eventually even be able to use "gudajsdie" in a manner consistent with its appearance—all without being able to understand "gudajsdie" in terms of "real" words. This is relative learning and essentially what Grosvenor sees as it combs the Internet. Grosvenor runs text through the Link Grammar System, a natural language parser (NLP) developed at Carnegie Mellon University, and assembles new nodes in its network. Every new word, excepting contextual cues, is inserted as a new node, usually with a local concept map as shown in Figure 1.

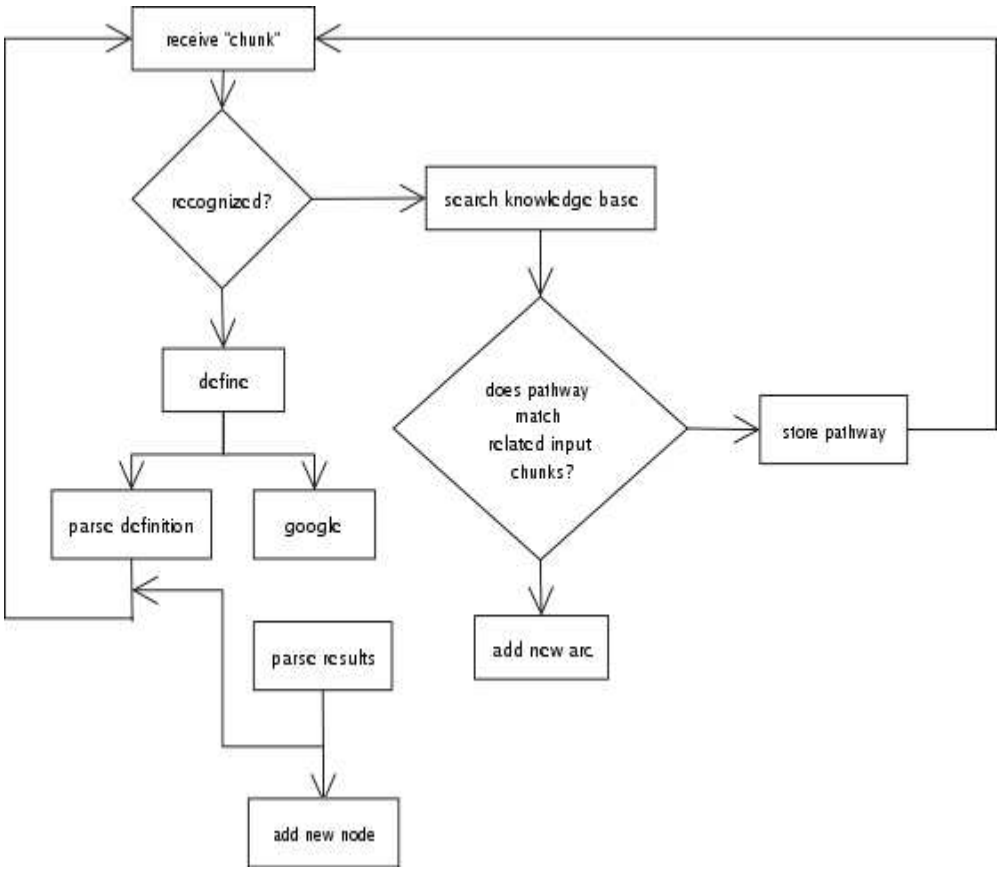

Figure 1

Philosophy

A number of philosophers have given arguments that strong AI is impossible, however, none have succeeded in presenting clear evidence that a human being is capable of having.

One of the problems in the philosophical arguments is the lack of clear distinction, if there even is any, between "weak" AI (the *imitation* of cognition) and "strong" AI, which Searle defines as "[the] computer really is a mind in the sense that ... can be literally said to understand and have other cognitive states" (Searle, 1980). "Cognitive states" may be taken to mean an actual change in the structure and functioning of a mind, as seen in brain scan images, whereas weak AI follows programmer-defined operations with little variance despite varying circumstances and input. Furthermore, such changes in the mental state should not be predefined or even reproducible by the programmer. According to Kleene (1967), such changes are impossible in a computer program as no computer program can improve on or explain all its own processes. To take Kleene's theory as an argument against AI requires proof that humans can derive their own internal functioning, but such ability has never been demonstrated. It may be argued that Kleene's argument regards the supposed inability of a program to surpass the capabilities and instructions provided by the programmer, but exceptions have been shown, for instance Harvey's experiments with reconfigurable hardware and genetic algorithms, in which the program is given only a goal and eventually achieves that goal without further

human intervention and even achieves the goal using techniques not fully understood by a human analysis of the final chip state (Harvey, 1999).

Wolfram proposes that any system can be recreated in a finite number of steps with a Turing machine (2002), so at some level even intelligence becomes an algorithm. No intelligence can overcome its physical foundations; the brain will always fundamentally be an electrical network, as will an AI. If Keene, Searle and others choose to take the physical nature of the brain for granted and argue on a level of some abstraction, so shall this paper. Assuming a computer, whether in hardware, software or combination of the two can provide a structure equivalent to that of the brain's "wetware", the question of intelligence now concerns the use of the fundamental structure.

Weak AI, such as Turing test capable chatbots, have set algorithms which process information as in Searle's Chinese box. One response to Searle's argument is that the individual may not understand the story but the system as a whole, of which the man is only one part, does. Searle rejoins that the individual may internalize all the system elements and still not understand Chinese and further, that such possibility creates the absurd consequence that "mind is everywhere". Searle's statement that it's ridiculous to say "that while [the] person doesn't understand Chinese, somehow the conjunction of that person and bits of paper might." With this statement, Searle seems to demand that *every* portion of a system be cognizant. Taken with the consideration that the brain is constructed of neurons, Searle is making the absurd demand that every neuron in your brain must individually understand English in order for you to be intelligent. Somewhere in the synapses intelligence arises.

The arguments against AI lead to two options: either intelligence contains some magical element and is impossible to replicate, or intelligence is much less complex and coherent that previously thought. The idea that intelligence contains some element that is irreproducible seems absurd and falls victim to Occam's razor.

Inescapably we are presented with two alternatives; either every neuron is individually capable of intelligence, there is something almost magic and inherently inducible about intelligence, or finally, and seemingly the most likely, intelligence is much less complex and coherent than ever imagined.

Young children have an astounding ability: they can learn about their world. Their ability is even more astounding when you consider they are born knowing so little, yet within a few years they have amassed a tremendous amount of knowledge through experiential learning (Kolb 1984). How can children understand anything when they don't have any base for bottom-up learning? Particularly, children can learn words and use them correctly without understanding or even knowing their definitions. According to the traditional bottom-up view of learning, such feats should be impossible; understanding B demands at least knowing A.

In Grosvenor's relative mental model, there is no need to know A before B because grokking is not necessary. Traditionally, knowing something implies grokking it, but in reality it is only necessary to know that B exists with relation to A. The actual individual meanings of A, B and any concepts connecting them are irrelevant and functionally non-

existent.

Relative comprehension and knowledge linking is Grosvenor's fundamental advantage. No single piece of knowledge (or agent) contains intelligence or understanding, only as a group is there comprehension (Minsky 1988). The greatest mystery of the learning process is how to provide prerequisites for new knowledge. For example, how do you teach someone with no knowledge of animals, or flying what a bird is, without explaining the numerous sub-concepts of animal, flying, and the sub-concepts required to explain animals and flying and so on? The tactic this project takes is that the sub-concepts are not important. Grosvenor can encounter “bird” and having no prior knowledge of “bird”, will accept it as a new building block.

Traditionally, learning has been assumed to be bottom-up, i.e. learning builds on prior knowledge. Grosvenor challenges this view with an outward approach to learning.

The storage model allows reconstruction (memory recall) along the lines of Minsky. Memory recall is indistinguishable from encountering an object for the first time or recognition. Recognition begins when an element of the object is recognized (the trigger element). That element activates one or more k-lines--stored paths between nodes that were formed in an earlier, similar experience. Computationally, these are represented as schema influencing a genetic algorithm which "spiders" through nodes connected to the trigger. This algorithm makes recursive passes through the node structures, adding more information which correlates with the perceived object on each pass, eventually producing a mental state representing the object. This mental state is, in most cases, indistinguishable from the state produced when the object was first experienced. This discrete storage technique provides for great flexibility and space efficiency in the memory system.

To illustrate with an example, imagine seeing a pencil. According to Grosvenor's cognitive model, you first recognize some distinctive feature, perhaps the shape or the material. Your mind then activates memory nodes that were active the last time you saw an object with similar characteristics and uses those linkages as guidelines to add increasingly more elements to your mental model, checking to make sure the elements are consistent with your visual input and eventually producing the symbol "pencil". This strategy saves you from having to store every possible variation of the pencil object and allows you to tentatively identify even the most unusual pencil, or use "pencil" in a comparison with, for instance, a pencil-sized object.

There have been many past attempts to create and artificial intelligence, ranging from simple chat programs such as Eliza and AIML to complex information systems such as Raphael's SIR (Minsky, 1968) and Cyc (Gutha 1998).

Cyc's stated goal was the development of a computer with the commonsense of a four year old child. The flaw in the Cyc project was trying to create a four year old that could be taught, but could not learn. Cyc approached the problem as though the computer had the high-level thought processes of a child in the concrete, or even formal operations, stage of development—the level of an adolescent. Worse, the Cyc project fed the computer knowledge from their perspective, struggling to cross reference every item

down to the most fundamental effort. Not only was this a waste of manpower, but it was completely ineffective. It is nearly useless to give a computer commonsense knowledge for two reasons.

First, a human being cannot adequately express commonsense knowledge and relate it to a computer. Commonsense knowledge is much more than just knowledge; it is a pattern of expectations which can only be learned through experience, not through teaching.

Secondly, most commonsense knowledge applies to the "real" world. A stand alone computer has no contact with the "real" world and cannot reinforce most commonsense knowledge. However, it is not entirely impossible that Grosvenor will develop commonsense. For example, Grosvenor may consistently encounter "down" in connection with falling objects and develop a "commonsense" idea of gravity. The development of commonsense is an overrated element of intelligence which, in a true intelligence, is dependent on the environment and experience. Cyc's classification of things as “TangibleObject” and “IntangibleObject” is a meaningless and artificial distinction for a computer program which resides exclusively in a world of the intangible.

The system proposed here is self-contained: it is capable of autonomously acquiring information regarding the world around it—the Internet world of information, if not the "real", physical world.

Grosvenor is an independent learning system capable of acquiring and interpreting knowledge independently of a human being, a key advantage over an expert system which generally requires a human expert to engage in lengthy Q&A sessions with the system. Dependence on humans was also Cyc's fundamental flaw. The Cyc project required human volunteers to scan information sources and decide how to explain information to a computer in terms of absolute knowledge, creating an interpretation limited by the human's own knowledge and perceptions of what the information should relate to (Guha & Lenat 1990).

Secondly, Grosvenor uses a building block approach to knowledge; each “block” ,or agent, is a single piece of knowledge that cannot be expressed (based on Grosvenor's current knowledge) as a combination of concepts. In other words, Grosvenor does not need to have absolute comprehension of a concept, only comprehension relative to other concepts.

Computational Model

One of the main points of this model is its ability to be implemented in a software infrastructure.

Grosvenor must obtain information in order to learn. To do so, it sorts through the Internet using the Google search engine (Brin & Page, 1998) and analyzes page content using a modified version of a natural language parser (NLP) developed at Carnegie Mellon. After dissecting the information with the NLP, it is then stored in a graph data

structure (essentially an adjacency-list graph). The use of a graph structure allows easy association and searching.

Using the W3C's libwww, an http request is made to google.com with the search query. This query returns a HTML document which is parsed on-the-fly by a SAX parser with callbacks activated by key elements in the HTML code. A typical 100 result page is about 90KB. The Google result page is broken down into a link and text description and stored in a data type. This result structure is then passed to the NLP which analyzes Similar requests are made to dictionary.com and thesaurus.com. Each request requires approximately 150KB of data to be transfered via Internet.

The program is multi-threaded and is initialized as three processes (NLP, database, Internet search and control) communicating via IPC. The multi-threaded model will ease any future transition to MPI and also minimizes the impact of data request bottlenecks on the program's runtime.

Google was used as the intermediary Internet search engine largely because of its PageRank system. The PageRank system provides search results that are more accurate and, in fact, incorporate a certain amount of intelligence in the result selection (Brin & Page 1998).
The prevent information loss and excessive RAM requirements, the program uses a PostgreSQL database for non-volatile information storage.

This is presently the source of one the major bugs in the program; the program has difficulty swapping data between RAM and SQL, using outdated pointers which lead to crashes. failing to update node pointers and trying to access invalid memory chunks.

To reduce search times, the database is alphabetically keyed, i.e. there are 26 separate databases. To further reduce access time, a cache algorithm is used to predict and pre-load rows.

Results
At this point the software is largely functional but quite unpolished and subject to frequent crashing, mostly relating to memory management problems. Based on a preliminary analysis, Grosvenor shows promise as a knowledge interpretation tool, superior Internet search engine and a functional implementation of the theory outlined in this paper. Unfortunately, the program could not reach a point of sufficient stability and functionality in the available time to allow for evaluation as an AI.

Practical Uses
Grosvenor is not just an exercise in cognitive theory; it has many immediate practical applications, regardless of the theory's validity. There is a developing trend in Internet search engine technology to provide smarter searches. Grosvenor's knowledge models and adaptive algorithms are ideal for providing extremely relevant and contextual search results. Grosvenor could also be used as a brain storming tool. Given a concept, Grosvenor could present a knowledge map, showing the interconnections between apparently unrelated ideas. This application could be extended in a more advanced program to the point where Grosvenor fulfills its namesake and becomes a "nexialist"—a

cross-disciplinary expert, drawing on an unimaginably large knowledge base to present innovative solutions unseen by PhD.'s trapped in the mindset of their own field.

Future goals:

- The addition of a symbolic logic system would further extend the program's abilities, adding a new dimension to its level of interpretation. Such a system could feasibly be implemented in C++, but PROLOG would be a more flexible and simpler choice.
- The program has several commercial applications in the realms of expert systems, knowledge bases, data mining and Internet search engines. Commercialization would provide funding for further research and development of the project as a theoretical concept and provide valuable testing experience in the real-world.
- A parallel or distributed implementation of the program would substantially increase scalability and speed.
- A physical interface would allow for a more accurate simulation of the mind. Studying how the program copes with interactions in the physical environment, I.e. through a robotic platform, would test its cognitive ability and provide the opportunity to develop "commonsense".

Acknowledgments
I would like to thank Dr. Marvin Minsky for his pioneering work in AI and inspiring me to pursue this project.

References

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine, http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm.

Chambers, T. C. et al. (2002). Circumscribing referential domains during real-time language comprehension. Journal of Memory and Language, 47, 30-49.

Chomsky, N. (1965). Aspects of the theory of syntax. Cambridge: MIT Press.

Chomsky, N. (1968). Language and mind. New York: Harcourt Brace.

Downs, R. M., & Stea, D., ed. (1973). Image and environment. Chicago: Aldine.

Geraci L., & Rajaram S. (2002) The orthographic distinctiveness on direct and indirect tests of memory: delineating the awareness and processing requirements. Journal of Memory and Language, 47, 273-291.

Guha, R. V., & Lenat., D. B. (1990). Cyc: a midterm Report. AI Magazine.

Johnson, et al. (2002) Text induction algorithm. IBM Systems Journal, 41.

Kleene, S.C. (2000). Mathematical logic. New York: Dover.

Kolb, D. (1984). Experiential learning. New Jersey: Prentice Hall.

Mattys, S. L., & Clark, J. H. (2002) Lexical activity in speech processing. Journal of Memory and Language, 47, 343-359.

Minsky, M. (2002). The emotion machine.
http://web.media.mit.edu/~minsky/E1/eb1.html

Minsky, M. (ed). (1969). Semantic information processing. Cambridge: MIT Press.

Minsky, M. (1988). The society of mind. New York: Touchstone Books.

Piaget, J. (1965). The child's conception of number. New York: Norton.

Piaget, J. (1966). The child's conception of physical causality. London: Routledge.

Russell, B. (1948). Human knowledge: its scope and limits. New York: Simon & Schuster.

Saffral, J. R. (2002). Constraints on statistical language learning. Journal of Memory and Language, 47, 172-196.

Searle, J. R. (1982), The Chinese Room Revisited, Behavioral and Brain Sciences, 5, 345-348.

Wagman, M. (1991). Artificial intelligence and human cognition. New York: Praeger Publishers.

Wolfram, S. (2002). A new kind of science. Champaign: Wolfram Media.

Appendix A-Source Code

```
/**************************************************************************
****
                googleHit.cpp  -  class definition and support for
google hits                     ------------------
   begin             : Mon Sep 23 2002
   copyright          : (C) 2002 by Roeland Hancock
   email             : rhancock@presencedesigns.net
 **************************************************************************
****/

/**************************************************************************
****
 *                                                *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
*
 **************************************************************************
****/
#include <string>
#include <list>
const string
queryString="http://www.google.com/search?num=100&lr=lang_en&as_ft=i&as_qdr=al
l&as_occt=any&as_dt=i&safe=images&as_q=";
class googleQuery {
public:
  googleQuery(string keywords);
  ~googleQuery;
  int addHit(char* summary, int rank);
  int addLink(char* link, int rank);
  int getHit(int rank);
  int getLink(int rank);
private:
  list<string> summaries;
};

int googleQuery::addHit(char* summary, int rank) {
}
int googleQuery::googleQuery(string keywords) {
                query=queryString+=keywords;
}

/**************************************************************************
****
                gparse.c (v.2) - extract summaries from google search results
takes argument of http://google.com/search?q=xxxxx

                  ------------------
   begin             : Wed Sep  11 20:25:02 MDT 2002 v0.1
   copyright          : (C) 2002 by Roeland Hancock/Presence Designs
```

```
   email                : rhancock@presencedesigns.net
   app                                  :Grosvenor
UPDATE 25.09.2002 17:43  v0.2   Add handling for non-html results. Add code to
request cached page if present
 ********************************************************************************
***/

#include <slist>
#include <string>
#include <string.h>

#include "WWWLib.h"
#include "WWWInit.h"
#include "WWWHTML.h"
typedef struct googleHit_s googleHit;
struct googleHit_s {
                int id;
                string url;
                string summary;
};
googleHit *currentHit;
currentHit=new googleHit;

slist <googleHit> gHitList;


BOOL hit=NO;
BOOL headerdone =NO;
BOOL divfound=NO;
int hitCount;
BOOL flTag=NO;

PRIVATE void addText (HText * text, const char * buf, int len)
{
  // char * bufcpy=buf;
if(hit) {

  if(buf && flTag && strstr(buf, "Cached")) printf("\nCached results on hit %d\n",
hitCount);
    if (buf) currentHit->summary.append(buf);
    flTag=NO;
  }
}

PRIVATE int printer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stdout, fmt, pArgs));
}
```

```
PRIVATE int tracer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stderr, fmt, pArgs));
}

PRIVATE int terminate_handler (HTRequest * request, HTResponse * response,
                                                void * param, int status)
{
   HTRequest_delete(request);

   HTProfile_delete();

   exit(0);
}

PRIVATE void beginElement (HText *                              text,
                                                int
                element_number,
                                                const BOOL *                    present,
                                                const char **           value)
{
   SGML_dtd * dtd = HTML_dtd();
   HTTag * tag = SGML_findTag(dtd, element_number);
   /*google precedes results with <div><!--m--><p>*/
if (tag) {
                char * tagname = HTTag_name(tag);
                int maxcnt = HTTag_attributes(tag);
                BOOL found = YES;
                if(!headerdone) {
                  if(divfound){
                    if(strcmp(tagname, "P") ==0){
                      headerdone=YES;
                    }
                    else divfound=NO;
                  }

                    else {
                      if(strcmp(tagname, "DIV") ==0) divfound=YES;

                  }
                }
                else {
                 /*look for cache tags (with CLASS=fl)*/
                  if(hitCount > 0) {
                    if(strcmp(tagname, "A")==0){
                    int cnt;
                    for (cnt=0; cnt<maxcnt; cnt++) {
```

```
      if (present[cnt]) {
                  char * attrname = HTTag_attributeName(tag, cnt);
                  if(strcmp(attrname, "CLASS")==0) {
                    if(strcmp(value[cnt], "fl")==0) flTag=YES;
                  }
      }

     }
     }
   }
     /*a hit starts with an <a href="...>  WITHOUT a class*/
   /*using XOR would make things cleaner*/
 if(!hit){
   hit=YES;
   if(strcmp(tagname, "A")==0){
     int cnt;
     for (cnt=0; cnt<maxcnt; cnt++) {
       if (present[cnt]) {
                  char * attrname = HTTag_attributeName(tag, cnt);
                  if(strcmp(attrname, "CLASS")==0) hit = NO;
       }
       if (hit) {
                  hitCount++;
                  HTPrint("---BEGIN HIT---\n");
       }
     }
   }
 }
 else {
   /*currently on a hit, stop at <a href=".." class=">*/
   if (strcmp(tagname, "A") ==0){
     int cnt;
     for (cnt=0;cnt<maxcnt; cnt++) {
       if (present[cnt]) {
                  char * attrname = HTTag_attributeName(tag, cnt);
                  if(strcmp(attrname, "CLASS")==0){
                    hit=NO;
                    HTPrint("---END HIT---\n");
                  }

       }
     }
   }
 }

 }
 }
```

```
}


 PRIVATE void endElement (HText * text, int element_number)
{
   /*libwww requires this handler, at the moment we dont care about end tags*/
}

/*dont need un parsed elemnet handlers either-google is clean html*/
PRIVATE void unparsedBeginElement (HText * text, const char * buf, int len)
{

}

PRIVATE void unparsedEndElement (HText * text, const char * buf, int len)
{


}

int main (int argc, char ** argv)
{
   char * uri = NULL;

   /* init libwww, setup handlers*/
   HTProfile_newHTMLNoCacheClient ("gparse", "0.1");
   HTPrint_setCallback(printer);
   HTTrace_setCallback(tracer);
   HTNet_addAfter(terminate_handler, NULL, NULL, HT_ALL, HT_FILTER_LAST);

   HText_registerElementCallback(beginElement, endElement);
   HText_registerUnparsedElementCallback(unparsedBeginElement,
                                                                  unparsedEndElement);
   HText_registerTextCallback(addText);

#if 0
   HTSetTraceMessageMask("sop");
#endif
   HTAlert_setInteractive(NO);

   /* timeout is 15sec*/
   HTHost_setEventTimeout(15000);


   if (argc >= 2)
                uri = HTParse(argv[1], NULL, PARSE_ALL);
```

```
   if (uri) {
                HTRequest * request = NULL;
                HTAnchor * anchor = NULL;
                BOOL status = NO;

                request = HTRequest_new();

                anchor = HTAnchor_findAddress(uri);
                status = HTLoadAnchor(anchor, request);
                if (status == YES) HTEventList_loop(request);

   } else {
                HTPrint("Type
http://www.google.com/search?as_q=XXXXXX&num=100&lr=lang_en&as_ft=i&as_qd
r=all&as_occt=any&as_dt=i&safe=images\n");
   }

   return 0;
}
/***************************************************************************
****
                graph.cpp  -  description
                  -------------------
   begin             : Mon Oct 21 2002
   copyright          : (C) 2002 by Roeland Hancock
   email             : rhancock@presencedesigns.net
 ***************************************************************************
****/

/***************************************************************************
****
 *                                                                         *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
*
 ***************************************************************************
****/
struct node {
                node *next;
                node *prev;
                string concept;
                part_of_speech pos;
                bool visited;
                int weight;
};
build_graph(node& parent) {
node *current_node;
node *last_node;
current_node=parent;
```

```
last_node=current_node;
vector<*node> children;
while(current_node->next) {
                children.push_back(current_node->next);
                current_node=current_node->next;
}
for (int i=0; i < children.size(); i++){
                build_graph(children[i]);
}
#include <deque>
#include <string>
#include <string.h>
#define MAX_MAP_ELEMENTS 2048
#define MAX_MAP_OVERFLOW 255

class nodeToStringMap {
protected:
  vector<deque <string> > conceptMap (26);
  vector<deque <*node> > nodeMap (26);



public:
  nodeToStringMap();
  ~nodeToStringMap();
  //int addMap(nodeId id, string concept);
  int addMap(node *n, string concept);
  *node lookupMap(string concept);
  int charToInt(char c);


};

inline nodeToStringMap::nodeToStringMap() {
  //use deques
  new deque <string> conceptMap;
  new deque <&node> nodeMap;
  new deque<bool> isLoaded;
}
inline nodeToStringMap::~nodeToStringMap() {
  delete conceptMap;
  delete nodeMap;
  delete isLoaded;
}

inline nodeToStringMap::loadKnowledgeToMemSave() {

}
```

```
inline nodeToStringMap::loadKnowledgeToMem() {

}
inline nodeToStringMap::offloadKnowledgeSave() {

}

inline nodeToStringMap::purgeMap() {

}

inline int nodeToStringMap::addMap(node *n, string concept, int index){
  if (conceptMap[index]->size() > MAX_MAP_ELEMENTS +
MAX_MAP_OVERFLOW) {
    offloadKnowledgeSave(MAX_MAP_OVERFLOW, index);
  }
  conceptMap[index]->push_back(concept);
  nodeMap.push_back(&node);
  if(conceptMap[index]->end() && nodeMap[index]->end()) {
    return conceptMap[index]->.end();
  }
  else return -1;
}

inline int  nodeToStringMap::delMap(node *node) {

}

inline nodeToStringMap::lookup() (deque<string>::iterator &itr, string concept) {
  itr=find(conceptMap[index]->begin(), conceptMap[index]->end(), concept);
  if(itr==conceptMap[index]->end()) {
    if (loadKnowledgeToMem(concept)<0) itr=NULL;
  }

}



inline char nodeToStringMap::charToInt(char c) {
  //ascii, everything should be lowercase
  //a=96, z=122
  return (int(c)-96);
}
#include <iostream.h>
#include <string>
#include <vector.h>
string t;
```

```
vector <int> v;


typedef struct arc_s arc;
typedef struct node_s node;
struct arc_s {
  node * nextNode;
  int weight;
};

struct node_s {
  node * nextNode;
  char * concept;
  vector <arc> arcs;
  int visited;
};


node * root;
node * addedNode;
root->concept="rootnode";
string c="im a node";
/*addNode( node * nd, arc * vert) {
  nd->arcs.push_back(*vert);

}
*/


node * addNodeArc (string strconcept, node * nd2, int wt) {
  node * newNode;
  arc * newArc;
  newArc->weight=wt;
  newArc->nextNode=nd2;
  newNode->concept=strconcept;
  newNode->arcs.push_back(newArc);
  newArc->nextNode=newNode;
  nd2->nextNode.push_back(newArc);
  return newNode;
}
addedNode=addNodeArc(c, root, -3);
for (int i=0; i<10; i++) {
  c+="a";
  addedNode=addNodeArc(c, addedNode, i);
}
/**************************************************************************
****
                    main.cpp  -  description
```

```
                  -------------------
   begin             : Mon Sep 30 16:05:49 MDT 2002
   copyright         : (C) 2002 by Roeland Hancock
   email             : rhancock@presencedesigns.net
 ***************************************************************************
****/

/***************************************************************************
****
 *                                                                         *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
*
 ***************************************************************************
****/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  cout << "Hello, World!" << endl;

  return EXIT_SUCCESS;
}
/***************************************************************************
****
                    misc.cpp  -  miscellanous support functions
                      -------------------
   begin             : Mon Sep 30 2002
   copyright         : (C) 2002 by Roeland Hancock
   email             : rhancock@presencedesigns.net
 ***************************************************************************
****/

/***************************************************************************
****
 *                                                                         *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
*
 ***************************************************************************
****/
#include <string>
#include <string.h>
//map a non url safe character to html %0x##
```

```
void htmlSafeString(string &unsafeURL) {
		string unsafeChars("$&+,/:;=?@ \"<>#%{}|\\^~[]`"); /*WARNING!! check
if these
		chars are allowed in strings if this doesnt compile*/
		string
safeChars("24262b2c2f3a3b3d3f4020223c3e23257b7d7c5c5e7e5b5d60");
		int pos=0;
		int pos2=0;
		while(pos!=npos ||pos<unsafeChars.length()){
			pos=unsafeURL.find_first_of(unsafeChars, pos);
			if(pos!=npos && pos!=0) {
				pos2=unsafeChars.find(unsafeURL[pos]);
				unsafeChars.replace(pos,1,"%");
				unsafeURL.insert(++pos, safeChars, pos2, 2);
				pos++;
			}
		}
 }/**************************************************************************
******
		nlp.cpp  -  description
		  -------------------
   begin		: Mon Sep 30 2002
   copyright		  : (C) 2002 by Roeland Hancock
   email		: rhancock@presencedesigns.net
 **************************************************************************
****/

/**************************************************************************
****
 *                                                                        *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
*
 **************************************************************************
****/
#include "nlp/link-includes.h"

int linkParse() {

		typedef struct CNode_s CNode;
		struct CNode_s {
			char  * label;
			CNode * child;
			CNode * next;
			int   start, end;
		};

 CNode=linkage_constituent_tree(Linkage linkage);
   Dictionary    dict;
```

```
    Parse_Options opts;
    Sentence      sent;
    Linkage       linkage;
    char *        diagram;
    int           i, num_linkages;
    char *        input_string[] = {};

    opts  = parse_options_create();
    dict  = dictionary_create("4.0.dict", "4.0.knowledge", NULL, "4.0.affix");


    for (i=0; i<2; ++i) {
                sent = sentence_create(input_string[i], dict);
                num_linkages = sentence_parse(sent, opts);
                if (num_linkages > 0) {
                   linkage = linkage_create(0, sent, opts);
                   printf("%s\n", diagram = linkage_print_diagram(linkage));
                   string_delete(diagram);
                   linkage_delete(linkage);
                }
                sentence_delete(sent);
    }

    dictionary_delete(dict);
    parse_options_delete(opts);
    return 0;
}
#include "link-includes.h"

/*int main() {

    Dictionary    dict;
    Parse_Options opts;
    Sentence      sent;
    Linkage       linkage;
    char *        diagram;
    int           i, num_linkages;
    char *        input_string[] = {
      "Grammar is useless because there is nothing to say -- Gertrude Stein.",
      "Computers are useless; they can only give you answers -- Pablo Picasso."};

    opts  = parse_options_create();
    dict  = dictionary_create("4.0.dict", "4.0.knowledge", NULL, "4.0.affix");

    for (i=0; i<2; ++i) {
                sent = sentence_create(input_string[i], dict);
                num_linkages = sentence_parse(sent, opts);
                if (num_linkages > 0) {
```

```
                linkage = linkage_create(0, sent, opts);
                printf("%s\n", diagram = linkage_print_diagram(linkage));
                string_delete(diagram);
                linkage_delete(linkage);
            }
            sentence_delete(sent);
    }
    dictionary_delete(dict);
    parse_options_delete(opts);
    return 0;
}
*/
/* Print out the words at the leaves of the tree,
   bracketing constituents labeled "PP" */

void print_words_with_prep_phrases_marked(CNode *n) {
    CNode * m;
    static char * spacer=" ";

    if (n == NULL) return;
    if (strcmp(n->label, "PP")==0) {
            printf("%s[", spacer);
            spacer="";
    }
    for (m=n->child; m!=NULL; m=m->next) {
            if (m->child == NULL) {
                printf("%s%s", spacer, m->label);
                spacer=" ";
            }
            else {
                print_words_with_prep_phrases_marked(m);
            }
    }
    if (strcmp(n->label, "PP")==0) {
            printf("]");
    }
}

int main() {

    Dictionary    dict;
    Parse_Options opts;
    Sentence      sent;
    Linkage       linkage;
    CNode *       cn;
    char *        string;
    char *        input_string =
      "This is a test of the constituent code in the API.";
```

```
    opts  = parse_options_create();
    dict  = dictionary_create("4.0.dict", "4.0.knowledge",
                                                "4.0.constituent-knowledge", "4.0.affix");

    sent = sentence_create(input_string, dict);
    if (sentence_parse(sent, opts)) {
                linkage = linkage_create(0, sent, opts);
                printf("%s", string = linkage_print_diagram(linkage));
                string_delete(string);
                cn = linkage_constituent_tree(linkage);
                print_words_with_prep_phrases_marked(cn);
                linkage_free_constituent_tree(cn);
                fprintf(stdout, "\n\n");
                linkage_delete(linkage);
    }
    sentence_delete(sent);

    dictionary_delete(dict);
    parse_options_delete(opts);
    return 0;
}

/*Find noun labels,
find link verbs
find objects
*/
/*************************************************************************
****
                    parsecontrol.cpp  -  interfaces with tparse.c and
                      gparse.c to extract results and pass to NLP
                       -------------------
    begin                : Fri Sep  13 23:32:02 MDT 2002
    copyright            : (C) 2002 by Roeland Hancock/Presence Designs
    email                : rhancock@presencedesigns.net
    app                                        :Grosvenor
 *************************************************************************
****/
#include <iostream>
#include <string>
#include <vector>
#include "googleHit.h"
const int MAX_GOOGLE_HITS=100;
const int MAX_ROGETS_HITS=10;

int getGoogle(string searchTerms, int limit);
int addGoogleHit(); //call from c
```

```
int googleHit::addHit(char* summary, int rank) {




int getGoogle(string searchTerms, int limit) {
  const char* query=searchTerms.data();
  //call c func
}
/**************************************************************************
****
                    pqwrite.cpp  -  write information to psql
                      -------------------
    begin              : Fri Dec 13 2002
    copyright          : (C) 2002 by Roeland Hancock
    email              : rhancock@presencedesigns.net
 **************************************************************************
****/

/**************************************************************************
****
 *                                                        *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
*
 **************************************************************************
****/

#include <iostream.h>
#include <iomanip.h>
#include "libpq++.h"

int main() {
const char* dbName = "dbname=grosvenor_k user=ch070rxh";
PgDatabase data(dbName);
data.ExecCommandOk("BEGIN");
data.ExecCommandOk("DECLARE myportal CURSOR FOR select * from
grosvenor_k");
data.ExecTuplesOk("FETCH ALL in myportal");
int nFields = data.Fields();
                for (int i = 0; i < nFields; i++)
                                cout << setiosflags(ios::right) << setw(15) <<
data.FieldName(i);
                cout << endl << endl;

                // next, print out the instances
                for (int i = 0; i < data.Tuples(); i++)
```

```
		{
			for (int j = 0; j < nFields; j++)
				cout << setiosflags(ios::right) << setw(15) <<
data.GetValue(i, j);
			cout << endl;
		}

		// Close the portal
		data.Exec("CLOSE myportal");

		// End the transaction
		data.Exec("END");
		return 0;
}
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <string>
#include <queue>

pid_t searchProc;
int searchPipes[2];
pipe(searchPipes);
if((searchProc=fork())==0){
  //child
  queue<string> links;
  char buf[2000];
  void getNextQuery(int signo) {
    bytes=read(searchPipes[0],buf,2000 );
    string s(buf,bytes);
    links.push(s);
  }
  signal(SIGUSR1, getNextQuery);
}
else {
  //parent

}

/**************************************************************************
****
                soapserver.cpp  -  description
                  -------------------
    begin              : Wed Jan 1 2003
    copyright          : (C) 2003 by Roeland Hancock
    email              : rhancock@presencedesigns.net
 **************************************************************************
****/
```

```
/*************************************************************************
****
 *                                                                       *
 *   This code may be redistributed or modified under the terms of the BSD *
 *   License                                                             *
 *************************************************************************
****/

/*************************************************************************
****
                  gparse.h  -  description
                    -------------------
   begin             : Wed Oct 2 2002
   copyright           : (C) 2002 by Roeland Hancock
   email             : rhancock@presencedesigns.net
 *************************************************************************
****/

/*************************************************************************
****
 *                                                                       *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
                                                                         *
 *                                                                       *
 *************************************************************************
****/
#include "WWWLib.h"
#include "WWWInit.h"
#include "WWWHTML.h"

#ifndef _GPARSE_H
BOOL hit, headerdone, divfound;
int hitCount;
BOOL flTag;
PRIVATE void addText (HText * text, const char * buf, int len);
PRIVATE int printer (const char * fmt, va_list pArgs);
PRIVATE int tracer (const char * fmt, va_list pArgs);
PRIVATE int terminate_handler (HTRequest * request, HTResponse * response,
                               void * param, int status);
PRIVATE void beginElement (HText *                      text,
                           int
                element_number,
                           const BOOL *                 present,
                           const char **          value);
PRIVATE void unparsedBeginElement (HText * text, const char * buf, int len);
PRIVATE void unparsedEndElement (HText * text, const char * buf, int len);
#define _GPARSE_H
```

```
#endif/*************************************************************************
*********
                    soapserver.h  -  description
                      -------------------
   begin             : Wed Jan 1 2003
   copyright          : (C) 2003 by Roeland Hancock
   email             : rhancock@presencedesigns.net
 *************************************************************************
****/

/*************************************************************************
****
 *                                                 *
 *   This code may be redistributed or modified under the terms of the BSD *   *License
                                                  *
 *                                                 *
 *************************************************************************
****/

/*************************************************************************
****
                    gparse.c (v.1) - extract summaries from google search results
takes argument of http://google.com/search?q=xxxxx

                    -------------------
   begin             : Wed Sep  11 20:25:02 MDT 2002
   copyright          : (C) 2002 by Roeland Hancock/Presence Designs
   email             : rhancock@presencedesigns.net
   app                                  :Grosvenor
 *************************************************************************
****/


#include "WWWLib.h"
#include "WWWInit.h"
#include "WWWHTML.h"

BOOL hit=NO;
BOOL headerdone =NO;
BOOL divfound=NO;

PRIVATE void addText (HText * text, const char * buf, int len)
{
  if(hit) {

    if (buf) fwrite(buf, 1, len, stdout);
  }
}
```

```
PRIVATE int printer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stdout, fmt, pArgs));
}

PRIVATE int tracer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stderr, fmt, pArgs));
}

PRIVATE int terminate_handler (HTRequest * request, HTResponse * response,
                                                void * param, int status)
{
   HTRequest_delete(request);

   HTProfile_delete();

   exit(0);
}

PRIVATE void beginElement (HText *                              text,
                                                int
                element_number,
                                                const BOOL *                    present,
                                                const char **           value)
{
   SGML_dtd * dtd = HTML_dtd();
   HTTag * tag = SGML_findTag(dtd, element_number);
   /*google precedes results with <div><!--m--><p>*/
if (tag) {
                char * tagname = HTTag_name(tag);
                int maxcnt = HTTag_attributes(tag);
                BOOL found = YES;
                if(!headerdone) {
                  if(divfound){
                    if(strcmp(tagname, "P") ==0){
                      headerdone=YES;
                    }
                    else divfound=NO;
                  }

                    else {
                      if(strcmp(tagname, "DIV") ==0) divfound=YES;

                  }
                }
                else {
```

```
                /*a hit starts with an <a href="...>  WITHOUT a class*/
                /*using XOR would make things cleaner*/

              if(!hit){
                hit=YES;
                if(strcmp(tagname, "A")==0){
                  int cnt;
                  for (cnt=0; cnt<maxcnt; cnt++) {
                    if (present[cnt]) {
                              char * attrname = HTTag_attributeName(tag, cnt);
                              if(strcmp(attrname, "CLASS")==0) hit = NO;
                    }
                    if (hit) {
                              HTPrint("---BEGIN HIT---\n");
                    }
                  }
                }
              }
              else {
                /*currently on a hit, stop at <a href=".." class=">*/
                if (strcmp(tagname, "A") ==0){
                  int cnt;
                  for (cnt=0;cnt<maxcnt; cnt++) {
                    if (present[cnt]) {
                              char * attrname = HTTag_attributeName(tag, cnt);
                              if(strcmp(attrname, "CLASS")==0){
                                hit=NO;
                                HTPrint("---END HIT---\n");
                              }

                    }
                  }
                }
              }

              }
   }


}

 PRIVATE void endElement (HText * text, int element_number)
{
   /*libwww requires this handler, at the moment we dont care about end tags*/
}

/*dont need un parsed elemnet handlers either-google is clean html*/
PRIVATE void unparsedBeginElement (HText * text, const char * buf, int len)
```

```
{

}

PRIVATE void unparsedEndElement (HText * text, const char * buf, int len)
{


}

int main (int argc, char ** argv)
{
   char * uri = NULL;

   /* init libwww, setup handlers*/
   HTProfile_newHTMLNoCacheClient ("gparse", "0.1");
   HTPrint_setCallback(printer);
   HTTrace_setCallback(tracer);
   HTNet_addAfter(terminate_handler, NULL, NULL, HT_ALL, HT_FILTER_LAST);

   HText_registerElementCallback(beginElement, endElement);
   HText_registerUnparsedElementCallback(unparsedBeginElement,
                                                                  unparsedEndElement);
   HText_registerTextCallback(addText);

#if 0
   HTSetTraceMessageMask("sop");
#endif
   HTAlert_setInteractive(NO);

   /* timeout is 15sec*/
   HTHost_setEventTimeout(15000);


   if (argc >= 2)
                uri = HTParse(argv[1], NULL, PARSE_ALL);

   if (uri) {
                HTRequest * request = NULL;
                HTAnchor * anchor = NULL;
                BOOL status = NO;

                request = HTRequest_new();

                anchor = HTAnchor_findAddress(uri);
                status = HTLoadAnchor(anchor, request);
                if (status == YES) HTEventList_loop(request);
```

```
   } else {
                HTPrint("Type
http://www.google.com/search?as_q=XXXXXX&num=100&lr=lang_en&as_ft=i&as_qd
r=all&as_occt=any&as_dt=i&safe=images\n");
   }

   return 0;
}
/*****************************************************************************
                    gparse.c (v.1) - extract summaries from google search results
takes argument of http://google.com/search?q=xxxxx

                    -------------------
   begin             : Wed Sep  11 20:25:02 MDT 2002
   copyright          : (C) 2002 by Roeland Hancock/Presence Designs
   email             : rhancock@presencedesigns.net
   app                                    :Grosvenor
 *****************************************************************************
****/


#include "WWWLib.h"
#include "WWWInit.h"
#include "WWWHTML.h"

BOOL hit=NO;
BOOL headerdone =NO;
BOOL divfound=NO;

PRIVATE void addText (HText * text, const char * buf, int len)
{
  if(hit) {

   if (buf) fwrite(buf, 1, len, stdout);
  }
}

PRIVATE int printer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stdout, fmt, pArgs));
}

PRIVATE int tracer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stderr, fmt, pArgs));
}
```

```
PRIVATE int terminate_handler (HTRequest * request, HTResponse * response,
                                                void * param, int status)
{
   HTRequest_delete(request);

   HTProfile_delete();

   exit(0);
}

PRIVATE void beginElement (HText *                              text,
                                                int
                element_number,
                                                const BOOL *                    present,
                                                const char **           value)
{
   SGML_dtd * dtd = HTML_dtd();
   HTTag * tag = SGML_findTag(dtd, element_number);
   /*google precedes results with <div><!--m--><p>*/
if (tag) {
                char * tagname = HTTag_name(tag);
                int maxcnt = HTTag_attributes(tag);
                BOOL found = YES;
                if(!headerdone) {
                  if(divfound){
                    if(strcmp(tagname, "P") ==0){
                      headerdone=YES;
                    }
                    else divfound=NO;
                  }

                    else {
                      if(strcmp(tagname, "DIV") ==0) divfound=YES;

                  }
                }
                else {
                  /*a hit starts with an <a href="...>  WITHOUT a class*/
                  /*using XOR would make things cleaner*/

                if(!hit){
                  hit=YES;
                  if(strcmp(tagname, "A")==0){
                    int cnt;
                    for (cnt=0; cnt<maxcnt; cnt++) {
                      if (present[cnt]) {
                                char * attrname = HTTag_attributeName(tag, cnt);
                                if(strcmp(attrname, "CLASS")==0) hit = NO;
```

```
                    }
                    if (hit) {
                            HTPrint("---BEGIN HIT---\n");
                    }
                   }
                  }
                }
                else {
                  /*currently on a hit, stop at <a href=".." class=">*/
                  if (strcmp(tagname, "A") ==0){
                    int cnt;
                    for (cnt=0;cnt<maxcnt; cnt++) {
                      if (present[cnt]) {
                            char * attrname = HTTag_attributeName(tag, cnt);
                            if(strcmp(attrname, "CLASS")==0){
                              hit=NO;
                              HTPrint("---END HIT---\n");
                            }

                      }
                    }
                  }
                }

                }
   }


}

 PRIVATE void endElement (HText * text, int element_number)
{
   /*libwww requires this handler, at the moment we dont care about end tags*/
}

/*dont need un parsed elemnet handlers either-google is clean html*/
PRIVATE void unparsedBeginElement (HText * text, const char * buf, int len)
{

}

PRIVATE void unparsedEndElement (HText * text, const char * buf, int len)
{


}

int main (int argc, char ** argv)
```

```
{
   char * uri = NULL;

   /* init libwww, setup handlers*/
   HTProfile_newHTMLNoCacheClient ("gparse", "0.1");
   HTPrint_setCallback(printer);
   HTTrace_setCallback(tracer);
   HTNet_addAfter(terminate_handler, NULL, NULL, HT_ALL, HT_FILTER_LAST);

   HText_registerElementCallback(beginElement, endElement);
   HText_registerUnparsedElementCallback(unparsedBeginElement,
                                                                  unparsedEndElement);
   HText_registerTextCallback(addText);

#if 0
   HTSetTraceMessageMask("sop");
#endif
   HTAlert_setInteractive(NO);

   /* timeout is 15sec*/
   HTHost_setEventTimeout(15000);


   if (argc >= 2)
                uri = HTParse(argv[1], NULL, PARSE_ALL);

   if (uri) {
                HTRequest * request = NULL;
                HTAnchor * anchor = NULL;
                BOOL status = NO;

                request = HTRequest_new();

                anchor = HTAnchor_findAddress(uri);
                status = HTLoadAnchor(anchor, request);
                if (status == YES) HTEventList_loop(request);

   } else {
                HTPrint("Type http://google.com/search?q=xxx\n");
   }

   return 0;
}
/*****************************************************************************
****
                        gparse.c (v.2) - extract summaries from google search results
takes argument of http://google.com/search?q=xxxxx
```

```
                    -------------------
   begin            : Wed Sep  11 20:25:02 MDT 2002 v0.1
   copyright          : (C) 2002 by Roeland Hancock/Presence Designs
   email            : rhancock@presencedesigns.net
   app                                    :Grosvenor
UPDATE 25.09.2002 17:43  v0.2   Add handling for non-html results. Add code to
request cached page if present
 ***************************************************************************
***/

#include <slist>
#include <string>
#include <string.h>

#include "WWWLib.h"
#include "WWWInit.h"
#include "WWWHTML.h"
typedef struct googleHit_s googleHit;
struct googleHit_s {
                int id;
                string url;
                string summary;
};
googleHit currentHit;

slist <googleHit> gHitList;


BOOL hit=NO;
BOOL headerdone =NO;
BOOL divfound=NO;
int hitCount;
BOOL flTag=NO;

PRIVATE void addText (HText * text, const char * buf, int len)
{
  // char * bufcpy=buf;
if(hit) {

  if(buf && flTag && strstr(buf, "Cached")) printf("\nCached results on hit %d\n",
hitCount);
    if (buf) fwrite(buf, 1, len, stdout);
    flTag=NO;
  }
}

PRIVATE int printer (const char * fmt, va_list pArgs)
{
```

```
   return (vfprintf(stdout, fmt, pArgs));
}

PRIVATE int tracer (const char * fmt, va_list pArgs)
{
   return (vfprintf(stderr, fmt, pArgs));
}

PRIVATE int terminate_handler (HTRequest * request, HTResponse * response,
                               void * param, int status)
{
   HTRequest_delete(request);

   HTProfile_delete();

   exit(0);
}

PRIVATE void beginElement (HText *                      text,
                           int
                element_number,
                           const BOOL *                 present,
                           const char **                value)
{
   SGML_dtd * dtd = HTML_dtd();
   HTTag * tag = SGML_findTag(dtd, element_number);
   /*google precedes results with <div><!--m--><p>*/
if (tag) {
                char * tagname = HTTag_name(tag);
                int maxcnt = HTTag_attributes(tag);
                BOOL found = YES;
                if(!headerdone) {
                  if(divfound){
                    if(strcmp(tagname, "P") ==0){
                      headerdone=YES;
                     }
                    else divfound=NO;
                  }

                    else {
                      if(strcmp(tagname, "DIV") ==0) divfound=YES;

                  }
                }
                else {
                 /*look for cache tags (with CLASS=fl)*/
                  if(hitCount > 0) {
                    if(strcmp(tagname, "A")==0){
```

```
    int cnt;
    for (cnt=0; cnt<maxcnt; cnt++) {
      if (present[cnt]) {
                    char * attrname = HTTag_attributeName(tag, cnt);
                    if(strcmp(attrname, "CLASS")==0) {
                      if(strcmp(value[cnt], "fl")==0) flTag=YES;
                    }
      }

    }
    }
  }
    /*a hit starts with an <a href="...>  WITHOUT a class*/
  /*using XOR would make things cleaner*/
if(!hit){
  hit=YES;
  if(strcmp(tagname, "A")==0){
    int cnt;
    for (cnt=0; cnt<maxcnt; cnt++) {
      if (present[cnt]) {
                    char * attrname = HTTag_attributeName(tag, cnt);
                    if(strcmp(attrname, "CLASS")==0) hit = NO;
      }
      if (hit) {
                    hitCount++;
                    HTPrint("---BEGIN HIT---\n");
      }
    }
  }
}
else {
  /*currently on a hit, stop at <a href=".." class=">*/
  if (strcmp(tagname, "A") ==0){
    int cnt;
    for (cnt=0;cnt<maxcnt; cnt++) {
      if (present[cnt]) {
                    char * attrname = HTTag_attributeName(tag, cnt);
                    if(strcmp(attrname, "CLASS")==0){
                      hit=NO;
                      HTPrint("---END HIT---\n");
                    }

      }
    }
  }
}

}
```

```
                }


}


 PRIVATE void endElement (HText * text, int element_number)
{
   /*libwww requires this handler, at the moment we dont care about end tags*/
}

/*dont need un parsed elemnet handlers either-google is clean html*/
PRIVATE void unparsedBeginElement (HText * text, const char * buf, int len)
{

}

PRIVATE void unparsedEndElement (HText * text, const char * buf, int len)
{


}

int main (int argc, char ** argv)
{
   char * uri = NULL;

   /* init libwww, setup handlers*/
   HTProfile_newHTMLNoCacheClient ("gparse", "0.1");
   HTPrint_setCallback(printer);
   HTTrace_setCallback(tracer);
   HTNet_addAfter(terminate_handler, NULL, NULL, HT_ALL, HT_FILTER_LAST);

   HText_registerElementCallback(beginElement, endElement);
   HText_registerUnparsedElementCallback(unparsedBeginElement,
                                                                  unparsedEndElement);
   HText_registerTextCallback(addText);

#if 0
   HTSetTraceMessageMask("sop");
#endif
   HTAlert_setInteractive(NO);

   /* timeout is 15sec*/
   HTHost_setEventTimeout(15000);


   if (argc >= 2)
```

```
                uri = HTParse(argv[1], NULL, PARSE_ALL);

    if (uri) {
                HTRequest * request = NULL;
                HTAnchor * anchor = NULL;
                BOOL status = NO;

                request = HTRequest_new();

                anchor = HTAnchor_findAddress(uri);
                status = HTLoadAnchor(anchor, request);
                if (status == YES) HTEventList_loop(request);

    } else {
                HTPrint("Type
http://www.google.com/search?as_q=XXXXXX&num=100&lr=lang_en&as_ft=i&as_qd
r=all&as_occt=any&as_dt=i&safe=images\n");
    }

    return 0;
}
#include "link-includes.h"
#include <string>
#include "graph.h"

/*  Many thanks to the authors (Daniel Sleator, David Temperley, and John Lafferty)
   for creating and allowing the non-commerical */
/*  use of this software (with source!) */


/* nlp.cpp */
/* Roeland Hancock, Presence Designs, rhancock@presencedesigns.net */
/* parse a block of text, pick out interesting words and build a local
   node */


int splitSentences(string &textblock, int &pos,  string key) {
  int bpos=pos;
  if((pos=textblock.find(key, pos))!=npos) {
    string textblock.substr(bpos, npos-bpos);
    char * parseblock=textblock.data();
    parse(parseblock);
    return 1;
  }
  else return 0;
}
void parse(char * text) {
```

```
void findNouns(CNode *n) {
   CNode * m;
   node * newLocalNode;

   if (n == NULL) return;
for (int c=0; c <18; c++) {
   if (strcmp(n->label, constituents[i])==0) {
                printf("%s[", spacer);
                spacer="";
   }
   for (m=n->child; m!=NULL; m=m->next) {
                if (m->child == NULL) {
                   printf("%s%s", spacer, m->label);
                   spacer=" ";
                }
                else {
                   print_words_with_prep_phrases_marked(m);
                }
   }
   if (strcmp(n->label, "PP")==0) {
                printf("]");
   }
}

int main() {

   Dictionary    dict;
   Parse_Options opts;
   Sentence      sent;
   Linkage       linkage;
   CNode *       cn;
   char *        string;

   opts  = parse_options_create();
   dict  = dictionary_create("4.0.dict", "4.0.knowledge",
                                                "4.0.constituent-knowledge", "4.0.affix");

   sent = sentence_create(input_string, dict);
   if (sentence_parse(sent, opts)) {
                linkage = linkage_create(0, sent, opts);
                string_delete(string);
                cn = linkage_constituent_tree(linkage);
                print_words_with_prep_phrases_marked(cn);
                linkage_free_constituent_tree(cn);
                fprintf(stdout, "\n\n");
                linkage_delete(linkage);
   }
   sentence_delete(sent);
```

```
    dictionary_delete(dict);
    parse_options_delete(opts);
    return 0;
}
 char * constituents(int i) {
 switch pos[i] {
   case 1:
   return "ADJP";
   case 2:
   return "SBAR";
   case 3:
   return "VP";
   case 4:
   return "QP";
   case 5:
   return "ADVP";
   case 6:
   return "SBAR";
   case 7:
   return "PP";
   case 8:
   return "QP";
   case 9:
   return "ADVP";
   case 10:
   return "PRT";
   case 11:
   return "NP";
   case 12:
   return "PP";
   case 13:
   return "SINV";
   case 14:
   return "S";
   case 15:
   return "VP";
   case 16:
   return "ADJP";
   case 17:
   return "VP";
   case 18:
   return "NP";
   case 0:
   return "VP";
 }
 }
/*************************************************************************
```

```
****
                    tparse.c (v.1) - get synonyms from thesaurus.com
takes argument of http:://www.thesaurus.com/cg-bin/search?config=roget&words=xxxx

                    -------------------
   begin            : Thur Sep  12 09:26:02 MDT 2002
   copyright          : (C) 2002 by Roeland Hancock/Presence Designs
   email            : rhancock@presencedesigns.net
   app                                    :Grosvenor
 ***************************************************************************
****/


#include "WWWLib.h"
#include "WWWInit.h"
#include "WWWHTML.h"

BOOL syn=NO;
BOOL t_strong=NO;

PRIVATE void addText (HText * text, const char * buf, int len)
{
  if(syn) {

    if (buf) fwrite(buf, 1, len, stdout);
  }
}

PRIVATE int printer (const char * fmt, va_list pArgs)
{
    return (vfprintf(stdout, fmt, pArgs));
}

PRIVATE int tracer (const char * fmt, va_list pArgs)
{
    return (vfprintf(stderr, fmt, pArgs));
}

PRIVATE int terminate_handler (HTRequest * request, HTResponse * response,
                                                    void * param, int status)
{
    HTRequest_delete(request);

    HTProfile_delete();

    exit(0);
}
```

```
PRIVATE void beginElement (HText *                              text,
                                        int
                element_number,
                                        const BOOL *                    present,
                                        const char **           value)
{
   SGML_dtd * dtd = HTML_dtd();
   HTTag * tag = SGML_findTag(dtd, element_number);
   /*a synonym follows <strong><a href="">*/
if (tag) {
                char * tagname = HTTag_name(tag);
                int maxcnt = HTTag_attributes(tag);
                BOOL found = YES;
                  if(t_strong){
                    if(strcmp(tagname, "A") ==0){
                     syn=YES;
                    }
                    else t_strong=NO;
                  }

                    else {
                      if(strcmp(tagname, "STRONG") ==0) t_strong=YES;

                  }
}
}

 PRIVATE void endElement (HText * text, int element_number)
{

 SGML_dtd * dtd = HTML_dtd();
   HTTag * tag = SGML_findTag(dtd, element_number);
   if(tag) {
     char * tagname = HTTag_name(tag);
     if(strcmp(tagname, "A")==0) syn=NO;
     if(strcmp(tagname, "STRONG")==0) t_strong=NO;


  }

}

/*dont need un parsed elemnet handlers either-google is clean html*/
PRIVATE void unparsedBeginElement (HText * text, const char * buf, int len)
{

}
```

```
PRIVATE void unparsedEndElement (HText * text, const char * buf, int len)
{


}

int main (int argc, char ** argv)
{
   char * uri = NULL;

   /* init libwww, setup handlers*/
   HTProfile_newHTMLNoCacheClient ("tparse", "0.1");
   HTPrint_setCallback(printer);
   HTTrace_setCallback(tracer);
   HTNet_addAfter(terminate_handler, NULL, NULL, HT_ALL, HT_FILTER_LAST);

   HText_registerElementCallback(beginElement, endElement);
   HText_registerUnparsedElementCallback(unparsedBeginElement,
                                                                  unparsedEndElement);
   HText_registerTextCallback(addText);

#if 0
   HTSetTraceMessageMask("sop");
#endif
   HTAlert_setInteractive(NO);

   /* timeout is 15sec*/
   HTHost_setEventTimeout(15000);


   if (argc >= 2)
                uri = HTParse(argv[1], NULL, PARSE_ALL);

   if (uri) {
                HTRequest * request = NULL;
                HTAnchor * anchor = NULL;
                BOOL status = NO;

                request = HTRequest_new();

                anchor = HTAnchor_findAddress(uri);
                status = HTLoadAnchor(anchor, request);
                if (status == YES) HTEventList_loop(request);

   } else {
                HTPrint("Type
http:://www.thesaurus.com/cgi-bin/search?config=roget&words=xxx\n");     }
   return 0;
```

```
}


/*
 * GrosVisual.java
 *
 * Created on February 5, 2003, 5:10 PM
 */

/**
 *
 * @author  rhancock
 */
public class GrosVisual extends javax.swing.JFrame {
      public String s="";
       public int i=0;
   /** Creates new form GrosVisual */
   public GrosVisual() {
      initComponents();

   }

   /** This method is called from within the constructor to
    * initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is
    * always regenerated by the Form Editor.
    */
   private void initComponents() {//GEN-BEGIN:initComponents
      jPanel1 = new javax.swing.JPanel();
      queryField = new javax.swing.JTextField();
      queryButton = new javax.swing.JButton();

      getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

      setName("graphArea");
      addWindowListener(new java.awt.event.WindowAdapter() {
         public void windowClosing(java.awt.event.WindowEvent evt) {
            exitForm(evt);
         }
      });

      queryField.setText("Enter a word.");
      queryField.setToolTipText("Ask!");
      jPanel1.add(queryField);

      queryButton.setText("Query");
      queryButton.addMouseListener(new java.awt.event.MouseAdapter() {
         public void mouseClicked(java.awt.event.MouseEvent evt) {
```

```
            queryButtonMouseClicked(evt);
        }
    });

    jPanel1.add(queryButton);

    getContentPane().add(jPanel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(110, 120, 180, 70));

    pack();
  }//GEN-END:initComponents

  private void queryButtonMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_queryButtonMouseClicked
    // Add your handling code here:
    i+=20;
    s=queryField.getText();
    repaint();
  }//GEN-LAST:event_queryButtonMouseClicked
  public void paint (java.awt.Graphics g) {
    g.drawString(s, i, i);
  }
  /** Exit the Application */
  private void exitForm(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_exitForm
    System.exit(0);
  }//GEN-LAST:event_exitForm

  /**
   * @param args the command line arguments
   */
  public static void main(String args[]) {
    new GrosVisual().show();
  }


  // Variables declaration - do not modify//GEN-BEGIN:variables
  private javax.swing.JTextField queryField;
  private javax.swing.JButton queryButton;
  private javax.swing.JPanel jPanel1;
  // End of variables declaration//GEN-END:variables

}
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GrosVisual extends JFrame {
```

```
    public GrosVisual2() {
                super ("Grosvenor Visualization Environment");

                queryField = new javax.swing.JTextField();
       queryButton = new javax.swing.JButton();
                queryField.setText("Enter a word.");
       queryField.setToolTipText("Ask!");
       queryButton.setText("Query");
       queryButton.addMouseListener(new java.awt.event.MouseAdapter() {
                                public void mouseClicked(java.awt.event.MouseEvent evt) {
                                   queryButtonMouseClicked(evt);
                                }
                   });


                getContentPane().add(queryField, BorderLayout.SOUTH);
                getContentPane().add(queryButton, BorderLayout.SOUTH);

                private void queryButtonMouseClicked(java.awt.event.MouseEvent evt) {
                   // Add your handling code here:
                   i+=20;
                   s=queryField.getText();
                   repaint();
                }
    }


import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

public class DomEcho
{
    static Document document;

    public static void main(String argv[])
    {
 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

```
try {
DocumentBuilder builder = factory.newDocumentBuilder();
  document = builder.parse( new File(argv[0]) );



    } catch (SAXException sxe) {
      // Error generated during parsing
      Exception  x = sxe;
      if (sxe.getException() != null)
          x = sxe.getException();
      x.printStackTrace();

    } catch (ParserConfigurationException pce) {
      // Parser with specified options can't be built
      pce.printStackTrace();

    } catch (IOException ioe) {
      // I/O error
      ioe.printStackTrace();
    }



import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import dom.wrappers.DOMParser;
import dom.DOMParserWrapper;

import java.io.*;
public class ChildNode {
    public int weight;
    public String concept=new String();
    public ChildNode(String c, int wt) {
                this.weight=wt;
                this.concept.c;
    }
}
public class ResultNode {
    public Vector children= new Vector();
    public int numChildren;
    public ResultNode() {
```

```
                this.numChildren=0;
    }
    public addChild(ChildNode child) {
                this.children.add(child);
                this.numChildren++;
    }

}
public class PrintUsingDOM
{
    boolean validation = true;
    DOMParserWrapper myDOMParser;
    String xmlFile;
    Document doc;

    public PrintUsingDOM(String fileName)
    {
        xmlFile = fileName;

    }

    public void print()
    {
        try
        {
          myDOMParser = (DOMParserWrapper) new dom.wrappers.DOMParser();

          //To validate or not
          myDOMParser.setFeature( "http://xml.org/sax/features/validation",validation );
          doc = myDOMParser.parse(xmlFile);
          walk(doc);
        }
        catch(Exception e)
        {
           System.out.println("Errors " + e);

        }

    }

    //walk the DOM tree and print as u go
    private void walk(Node node)
    {
        int type = node.getNodeType();
                NodeList childNodes;
                String elementName;
                Vector children=new Vector();
        switch(type)
```

```
    {

       case Node.ELEMENT_NODE:
       {
          elementName=node.getNodeName();
                              if(elementName=="node") {
                                 childNodes=node.getChildNodes();
                                 branches=childNodes.getLength();
                                 for (int branch=0; branch <branches; branch++) {
                                          node=node.getFirstChild();
                                          nnm=node.getAttributes(); //weight
                                          attr=(Attr)nnm.item(0);
                                          weight=Integer(attr.getNodeValue());
                                          node=node.getFirstChild();//a text node
                                          concept=getNodeValue();
                                          drawNode(concept, weight, branches, branch,
depth);
                                          node=node.getParentNode();
                                          node=node.getNextSibling(); //back at next child
node, ready to recurse
                              }
                              }

                                 conceptNodes[i]->concept=domNode.getNodeValue();
                              if(elementValue=="parent")
                                 conceptNodes[i]->parents[c]=getNodeValue();
                              if(elementName=="child")
                                 conceptNodes[i]->children[c]=getNodeValue();
                              if(elementName=="link")
                                 conceptNodes[i]->links[c]=getNodeValue();


          NamedNodeMap nnm = node.getAttributes();
          if(nnm != null )
          {
             int len = nnm.getLength() ;
             Attr attr;
             for ( int i = 0; i < len; i++ )
             {
                attr = (Attr)nnm.item(i);
                System.out.print(' '
                    + attr.getNodeName()
                    + "=\""
                    + attr.getNodeValue()
                    +  "" );
             }
          }
          System.out.print('>');
```

```
        break;

    }//end of element
    case Node.ENTITY_REFERENCE_NODE:
    {

      System.out.print('&' + node.getNodeName() + ';' );
      break;

    }//end of entity
    case Node.CDATA_SECTION_NODE:
    {
          System.out.print( "<![CDATA["
                + node.getNodeValue()
                + "]]>" );
           break;

    }
    case Node.TEXT_NODE:
    {
       System.out.print(node.getNodeValue());
       break;
    }
    case Node.PROCESSING_INSTRUCTION_NODE:
    {
       System.out.print("<?"
          + node.getNodeName() ) ;
       String data = node.getNodeValue();
       if ( data != null && data.length() > 0 ) {
          System.out.print(' ');
          System.out.print(data);
       }
       System.out.println("?>");
       break;

     }
}//end of switch


//recurse
for(Node child = node.getFirstChild(); child != null; child = child.getNextSibling())
{
   walk(child);
}

//without this the ending tags will miss
if ( type == Node.ELEMENT_NODE )
```

```
        {
            System.out.print("</" + node.getNodeName() + ">");
        }



    }//end of walk

    public static void main(String args[])
    {

        String xmlFileName="";
        if(args.length == 0)
        {
            //Check to ensure user XML file name to parse
            System.out.println("Usage::java PrintUsingDOM path/xmlFilename");
            System.exit(0);
        }
        else
        {
            xmlFileName = args[0];
        }

        PrintUsingDOM pud = new PrintUsingDOM(xmlFileName);
        pud.print();

    }//end of main

}//end of DOM
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Grosvenor extends JApplet {
    private JTextField queryField;
    public ask() {
                super("Welcome to Grosvenor");
                Container container = getContentPane();
                container.setLayout(new FlowLAyout());
                queryField=new JTextField(50);
                conatiner.add(queryField);
                queryHandler handler = new queryHandler();
                queryField.addActionListener(handler);
                setsize(640, 480);
                setVisible(true);
    }
    public static void main( String args[]) {
```

```
import org.apache.xerces.parsers.*;
import org.apahce.xerces.framework.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import java.io.*;
import java.util.*;
```