**BITS Pilani, Pilani Campus**
**2nd Sem. 2017-18**
**CS F211 Data Structures & Algorithms**
==========
**Lab X- (2nd Apr. to 7th Apr.)**
==========

**Topics**: Graphs: Representation and Elementary Operations

**Exercise 1**: Define the following (directed) graph operations:

- Graph createGraph(int numV) // create an empty graph with numV vertices
- Enumeration getAdjacent(Graph G, Vertex v) // get the vertices adjacent to v
- Graph addEdge(Graph G, Vertex v, Vertex vAdj) // add edge (v, vAdj)
- int degree(Graph G, Vertex v) // get the degree of v

Implement these operations separately using (i) adjacency matrix and (ii) adjacency list as representation. Define a suitable datatype Vertex and a function that maps a vertex value to an integer that can be used to index into an adjacency matrix or an adjacency list.

**[Expected Time: 15+15=30 minutes]**

**Exercise 2**: Implement a breadth-first-search procedure on directed graphs using the operations defined in Exercise 1.

**[Expected Time: 40 minutes]**

**Exercise 3**: Implement a toy-crawler that will build a document-graph starting with a very small root-set, crawling a collection of hyper-linked documents, and terminating when a certain depth is reached or the number of vertices exceeds a limit. The core engine will invoke bfs on roots, maintain a frontier set (of links to be explored), maintain a collection of documents visited, and test whether links extracted from visited documents are already visited and if not add them to the frontier set.

Use the *curl* library to access (i.e. download) the documents in the collection and to extract links. Test this toy-crawler on two different local (i.e. internal to BITS) collections

1. GNU library documentation [http://172.24.16.12/libc/index.html]
2. A sub-collection of Wikipedia [http://172.24.16.12/wiki/index.html]

Please ensure that you remove the temporary file (or, overwrite in same file) in which you are saving the remote URL (mylocalfile in webreads.c). Rules for parsing above collections:

1. Do not follow any external link
2. Stop at depth 10 from the root or when 200 nodes have been created, whichever happens earlier.
3. Only parse HTML files. Use href tag to extract next targets. Store URL as well as name in the Graph.
   <a href="URL"> Name </a> //Format of tags to be extracted
4. Some links might be broken. Point every broken link to a single sentinel node in the graph.

**[Expected Time: 45 mins. (for prototyping) + 2*15=30 mins. (for test runs) + 30 mins. (for tuning) = 105 minutes.]**

**Introduction to `libcurl`**

Curl [https://curl.haxx.se/] is command line tool and library for transferring data with URLs. It supports almost all popular network transfer protocols and offers a rich set of options for handling network data transfer. `libcurl` [https://curl.haxx.se/libcurl/] is a C library written over curl.

Typical work-flow of a URL downloading program written using libcurl:

1. Initialize curl global environment
   ```
   curl_global_init(CURL_GLOBAL_ALL);
   ```
2. Start a curl session
   ```
   CURL *curl_handle;
   curl_handle = curl_easy_init();
   ```
3. Set options in *easy-interface* of libcurl for downloading an URL. It is done using curl_easy_setopt(handle, option, value) function, where handle is handle of current culr session, option is the name/tag of the option being set, and value is the value of option to be set.

   ```
   curl_easy_setopt(curl_handle, CURLOPT_URL,
   "http://172.24.16.12/wiki/index.html");
   ```
   //sets URL to download

   ```
   curl_easy_setopt(curl_handle, CURLOPT_NOPROGRESS, 1L);
   ```
   //disables progress bar

   ```
   curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION,
   write_data);
   ```
   //specifies the name of the user defined C function (write_data) which will be called when the file at given URL has been downloaded.

   ```
   curl_easy_setopt(curl_handle, CURLOPT_WRITEDATA,
   webfile);
   ```
   //specifies the argument to be passed to function writing data after downloading the file. In this case, webfile should be a FILE pointer pointing an already open writable file.

4. Start the downloading
   ```
   curl_easy_perform(curl_handle);
   ```
5. Close and clean up curl session
   ```
   curl_easy_cleanup(curl_handle);
   ```

In a single curl session multiple URLs can be downloaded by iteratively setting appropriate options (Step 3) and downloading the URL (Step 4).

A sample source code (`webread.c`) has been given for demonstration. You may write your own program or modify it as per requirement. Ensure that you use (-lcurl) flag while compiling any code which uses libcurl.

```
gcc webread.c -lcurl -o mydownloader
./mydownloader
```