# CS 23200 Introduction to C and Unix

## Project 1: Crawling the Web (Due on **October 19**, 2023, Thursday, 11:59pm)

Although you will have more than two weeks to work on this project, please start early. If you get stuck, please feel free to come to office hours. I am here to help you, but I cannot help if you do not ask.

***READ THE ENTIRE PROJECT DESCRIPTION CAREFULLY BEFORE BEGINNING THE PROJECT.***

## Project goals:

This project will deepen your knowledge of **pointers** and **memory management** by giving you hands-on experience coding these concepts. You will also gain experience working with **self-referential structures** and **recursive functions** that operate on those data structures. Additionally, this project will increase your familiarity with the basics of C (e.g., loops, basic data types, arrays, etc.) and the development process on *nix machines.

This project is the first in an exciting sequence of projects that deal with exploring the web. These projects will build upon each other, so that in the end you will have built a simple search engine that works on real web data!

- This project involves the web crawler that jumps around to different web pages.
- The next project will involve processing the content of a web page.
- The final project will use the crawler and the content processor to build an index of web pages. That index will be used to find matches to web searches.

## Project description:

### *Work on a Web Crawler*

For this project, you will be completing a web crawler. The web crawler will start at some web page, follow a random link on that page, then follow a random link on that new page, and so on, keeping a list of the pages that it has visited. Your part of this project will be to complete the linked list data structure that stores the web addresses that the crawler visits. Note that this project deals with a **singly-linked list** (unlike the doubly-linked list we discussed in class). Singly-linked lists are simpler, because each node just points to the next node (i.e., no “previous” pointers). The last node in the list has a NULL next pointer to indicate the end. (FYI, the disadvantage with singly-linked lists is that you can’t traverse them backwards, while doubly-linked lists let you go forward and backward.)

The file **crawler.c** contains some code already written. You should only modify that file in the places that are marked "TODO: complete this". There are comments to help guide you through the code. Note that there are four functions that operate on the linked list that you need to complete: **contains**, **insertBack**, **printAddresses**, and **destroyList**. **Each of these functions must be recursive.** The comments indicate how those functions should behave. Only "printAddresses" should print any output to the terminal.

In Brightspace, you can find a zip file "project1_code.zip" that contains the following files:

- crawler.c: source C code for project 1
- getLinks.py: a help python code, which is called from crawler.c
- runTestCases.sh: a help shell script for testing the C code
- expected_results.txt: expected results after running "runTestCases.sh" in the Diamond server

### *Python Code*

The project uses the python code "getLinks.py". At the Diamond server, python3 has been installed. Please install python bs4 module in the Diamond server by running the following command:

```
$ python3 -m pip install bs4 --user
```

If you want to run the C code in a Windows machine, you need to install python3 and python bs4 module to the Windows. Moreover, if the executable file is "python.exe", instead of "python3.exe", you need to change line 160 in crawler.c file to use "python getLinks.py", instead of "python3 getLinks.py". To avoid such a confusion, it is recommended to use the Diamond server for the development.

### *Running the Program*

When completed, the program will take a web address and a number of hops from the command line and try to crawl the web for the given number of hops starting from the given address. It will print out the addresses that it found as it was crawling, along with some messages if it found a cycle or if it ran into a dead end. For example, if you run the following in a Linux server:

```
./crawler "https://www.pfw.edu" 5
```

you might see output like

```
seed = 1644604968
Dead end on hop 4: no outgoing links
https://www.pfw.edu
https://www.pfw.edu/chancellor/
https://www.pfw.edu/news-center
https://www.instagram.com/purduefortwayne/
```

Because the crawler follows random links, your output will probably not match this exactly. The randomness can make debugging difficult, because running the program different times will usually lead to different execution paths. To alleviate this problem, the program prints out "seed

= <some number>" at the beginning. If you add this seed as an additional command-line argument when running your program, like

```
./crawler "https://www.pfw.edu" 5 1644604968
```

then you should get exactly the same behavior as when previously using that seed (until the web pages change!). You can do this with any seed value in order to get the same behavior multiple times.

A couple other notes that arise from dealing with real web data:

- You might get errors from the Python script "getLinks.py" that parses the web pages (because they might be malformed or in an unexpected language). If so, just rerun your program to get a different random crawling trajectory.
- Because of the network traffic involved in fetching these pages, you should limit the number of hops to 50 or fewer when you are running your project.
- To help check your solutions, I have created a set of web pages that link to each other and will remain the same throughout this project. Here are some example traces, which should match your program's output exactly in the **Diamond** Linux server:

    -
      ```
      $ ./crawler "https://users.pfw.edu/chenz/testWeb/page_000001.html" 10 899
      seed = 899
      Cycle detected on hop 3: address https://users.pfw.edu/chenz/testWeb/page_000003.html
      Dead end on hop 5: no outgoing links
      https://users.pfw.edu/chenz/testWeb/page_000001.html
      https://users.pfw.edu/chenz/testWeb/page_000011.html
      https://users.pfw.edu/chenz/testWeb/page_000003.html
      https://users.pfw.edu/chenz/testWeb/page_000010.html
      https://users.pfw.edu/chenz/testWeb/page_000005.html
      ```

    -
      ```
      $ ./crawler "https://users.pfw.edu/chenz/testWeb/page_000010.html" 5 983
      seed = 983
      Dead end on hop 5: no outgoing links
      https://users.pfw.edu/chenz/testWeb/page_000010.html
      https://users.pfw.edu/chenz/testWeb/page_000021.html
      https://users.pfw.edu/chenz/testWeb/page_000004.html
      https://users.pfw.edu/chenz/testWeb/page_000028.html
      https://users.pfw.edu/chenz/testWeb/page_000000.html
      ```

    -
      ```
      $ ./crawler "https://users.pfw.edu/chenz/testWeb/page_000003.html" 4 187
      seed = 187
      Cycle detected on hop 2: address https://users.pfw.edu/chenz/testWeb/page_000028.html
      https://users.pfw.edu/chenz/testWeb/page_000003.html
      https://users.pfw.edu/chenz/testWeb/page_000028.html
      https://users.pfw.edu/chenz/testWeb/page_000013.html
      https://users.pfw.edu/chenz/testWeb/page_000004.html
      https://users.pfw.edu/chenz/testWeb/page_000002.html
      ```

To help you to test the crawler, two additional files have been provided: “runTestCases.sh” and “expected_results.txt”. Please upload your code and these files to the **Diamond** Linux server and try to run

```
$ chmod +x runTestCases.sh
$ ./runTestCases.sh
$ diff results.txt expected_results.txt
```

If there is no output from the last command, it means that the outputs from your program are identical to the outputs in “expected_results.txt”. Please note that running the same program under Windows and under Linux will give the **different** results for this project. Here “expected_results.txt” is **only** for the Diamond Linux server.

***To successfully complete the assignment***, you must complete the four linked-list functions in crawler.c so they are **recursive** functions that behave as the comments indicate. Your project should not have any memory leaks or other memory management errors. In the Diamond Linux server, use

```
$ valgrind —-leak-check=yes ./crawler ...
```

to check for leaks and other memory-management errors.

## Submission

Please upload the file crawler.c to your GitHub homework repository under “**project1**” directory.

**Grading rubric:**

- Code can be compiled – 20pt
- Implement function “contains” correctly – 30pt
- Implement function “insertBack” correctly – 60pt
- Implement function “printAddresses” correctly – 30pt
- Implement function “destroyList” correctly – 30pt
- No memory leaks and no memory errors from running valgrind – 20pt
- Coding style – 10 pt

Total: 200 points.

**Note**: If the grader cannot identify which function(s) causes the mistake, the grader will grade the project based on the number testing cases passed. Moreover, if the implemented function is not a recursive function, no point will be given for that function.

**Note**: If your code cannot be compiled, you cannot get any point for this project. Therefore, please make sure that your code can be compiled without any error or any warning.