COMP 2401 -- Assignment #3

Due: Friday, November 8, 2013 at 12:00 (noon)

Goal

You will modify your program from Assignment #2 to simulate a function call stack, implemented as a linked list of frames, using C in the Ubuntu Linux environment.

In this assignment, you will:

- define the data types and structures required to represent the function call stack as a linked list of frames
- write supporting functions to manage the function call stack
- modify the instrumentation functions to work with the call stack implemented as a linked list
- change the existing call stack output functions to print to a file instead of standard output

Learning Objectives

- perform more complex pointer manipulations by implementing a linked list
- work with dynamically allocated memory
- practice using double pointers in pass-by-reference parameter passing
- use file I/O function calls to output to a file
- **Note**: Of course, linked lists are not the best underlying data structure for representing a stack. The use of linked lists in this assignment is for exclusively pedagogical purposes.

Instructions

1. Data types

Modify the function call stack data type, so that the call stack is implemented as a **singly** linked list of frames, exactly as we saw in the "Advanced Linked Lists" section of the course notes:

- `StackType` corresponds to the function call stack and contains a pointer to the head of the linked list of frames
- `FrameNodeType` represents a node in the linked list
- your main function must define the function call stack as dynamically allocated memory
  - there must be only **one** instance of the function call stack in the entire program; do not make copies!
  - this is **not** a global variable

Notes:

- your program must use the definitions in the header file found here: a3Defs.h
- the header file declares a frame number in each frame; your program must initialize this value

2. Supporting stack functions

Implement the following support functions for the function call stack:

- `initStack` allocates and initializes the stack
- `push` adds a frame in LIFO order to the call stack
- `pop` removes from the stack the frame that was last added
- `cleanupStack` deallocates all the memory for the stack

Note: remember to manage your memory! do not leave any memory leaks

3. Instrumentation functions

Modify both `enterSumFunc` and `leaveSumFunc` functions to manipulate the function call stack as a linked list:

- `enterSumFunc` is called when a sum function begins; it must:
  - create a new dynamically allocated stack frame to store information about one of the sum functions
  - initialize all the frame data, as you did in Assignment #2
  - add the new frame to the function call stack
  - output the contents of the call stack, using the modified `dumpStack` function
- `leaveSumFunc` is called when a sum function returns; it must:
  - output the contents of the call stack, using the modified `dumpStack` function
  - remove the corresponding frame from the stack

Notes:

- your sum functions must call the instrumentation functions defined above
- the instrumentation functions **must** use the supporting `push` and `pop` functions

4. Stack output to file

Define, as a global variable, a pointer to the output file that will contain the contents of the call stack

- **remember!** global variables are to be used very rarely, this is a one-time exception!

Modify the call stack output functions to work with the linked list and output the contents of the stack to a file:

- `dumpStack` traverses the linked list of frames and outputs the contents to the output file
  - you must use the same output format as in Assignment #2
- `dumpVar` and `dumpBytes` must be modified to print to file as well

Notes:

- think about where you must open and close the output file!
- you must use the stack utility functions found here: a3Stack.c
- you can find some helpful sample code here: testUtil.c

Constraints

- **Design**:
  - you must separate your code into modular, reusable functions
  - **never** use global variables, unless otherwise instructed
  - compound data types **must** be passed by reference, not by value
  - you must manage your memory! use valgrind to find memory leaks
- **Reuse**:
  - you must include the given header file in your program and use its function prototypes exactly as defined
- **Implementation**:
  - your program must perform all basic error checking
  - it must be thoroughly commented
- **Execution**
  - programs that do not compile, do not execute, or violate any constraint are subject to severe deductions

Submission

You will submit in *cuLearn*, before the due date and time, **one** tar file that includes all the following:

- all source and header files
- a readme file, which must include:
  - a preamble (program author(s), purpose, list of source/header/data files)
  - exact compilation command(s)
  - launching and operating instructions

Grading

- **Marking breakdown:**

Component	Marks
data types	15
supporting stack functions	50
instrumentation functions	25
stack output to file	10

- **Assignment grade:**
  - Your grade will be computed based on the completeness of your implementation, plus bonus marks, minus deductions.
- **Deductions:**
  - **100 marks** if:
    - any files are missing from your submission, or if they are corrupt or in the wrong format
  - **50 marks** if:
    - the code does not compile using gcc in the Ubuntu Linux shell environment
    - unauthorized changes have been made to the header and/or source files provided for you
    - code cannot be tested because it doesn't run
  - **25 marks** if:
    - your submission consists of anything other than exactly one tar file
    - your program is not broken down into multiple reusable, modular functions
    - your code is not correctly separated into header and source files
    - your program uses global variables (unless otherwise explicitly permitted)
    - the readme file is missing or incomplete
  - **10 marks** for missing comments or other bad style (non-standard indentation, improper identifier names, etc)
- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features