COMP 2401 -- Assignment #2

Due: Friday, October 25, 2013 at 12:00 (noon)

Goal

You will write a program in C, in the Ubuntu Linux shell environment, to simulate the use and operation of a function call stack. Every time a function is called, a new stack frame is created to contain information about that function, and the stack frame is added to the function call stack in LIFO (last-in-first-out) order. When the function returns, the corresponding frame is removed from the call stack.

In this assignment, you will:

- define the data types and structures required to represent the function call stack; you will use arrays to represent all collections of data
- write a main function that prompts the user for a collection of integers and calls one of two given sum functions (one iterative and one recursive) to add the integers together
- modify the given sum functions to call instrumentation functions upon entry and return
- write two instrumentation functions: one that initializes and adds a frame to the function call stack when a sum function is entered, and one that removes a frame from the call stack when a sum function returns; both instrumentation functions output the contents of the call stack

Learning Objectives

- understand and modify existing code
- work with arrays and perform simple pointer manipulations
- get familiar with the basic operation of the function call stack mechanism
- compare the operation of the call stack using iterative and recursive function calls
- practice pass-by-value and pass-by-reference parameter passing
- use standard I/O function calls to interact with a user
- integrate basic error checking with user I/O

Instructions

1. Main and supporting functions

Write a main function and supporting functions that do the following:

- define the function call stack variable
  - 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
- prompt the user to input the values of an integer array; you can start by prompting for the number of integers
- prompt the user to select which sum function to call (iterative or recursive)
- call the appropriate sum function; you will use the code for the sum functions found here: a2Loop.c
- display the resulting sum to the user

Your program must use the definitions in the header file found here: a2Defs.h

2. Data types

Modify the given header file to define the following data types required to simulate the function call stack:

- `StackType` corresponds to the function call stack and holds the frames currently on the stack
- `FrameType` holds the data for a frame corresponding to a specific function; this includes the function name and information about its parameters
- `VarType` holds the information for a specific parameter

The data types that you define **must** work with the stack utility functions found here: a2Stack.c

3. Instrumenting the sum functions

Modify both `sumIterative` and `sumRecursive` functions (found in a2Loop.c) to call the instrumentation functions on entry and return. The sum functions must call `enterSumFunc` immediately upon entry to initialize a stack frame corresponding to the sum function. They must call `leaveSumFunc` when the sum function returns.

4. Instrumentation functions

Implement the two instrumentation functions:

```
void enterSumFunc(StackType *stkPtr, char *fname, int num, int *arr, int *sum)
void leaveSumFunc(StackType *stkPtr)
```

- `enterSumFunc` creates a new stack frame to store information about one of the sum functions, including its name (`sumIterative` or `sumRecursive`) and the parameters passed to it; `enterSumFunc` must:
  - initialize a new stack frame
  - set the frame's function name to the value in `fname`
  - initialize the frame's parameter data with information about the parameters passed to the sum function
    - `num` represents the number of elements to be added together by the sum function
    - `arr` is the array of integers to be added together
    - `sum` points to the result of the sum function
  - add the new frame as the next frame in LIFO sequence on the call stack pointed to by `stkPtr`
  - output the contents of the function call stack, using the given `dumpStack` function
- `leaveSumFunc` prints out the contents of the function call stack when one of the sum functions returns, using the given `dumpStack` function

Constraints

- **Design**:
  - you must separate your code into modular, reusable functions
  - **never** use global variables
  - compound data types **must** be passed by reference, not by value
- **Reuse**:
  - you must include the given header file in your program and use its function prototypes exactly as defined
  - you must use the sum functions found here: a2Loop.c
  - you must use, without modification, the stack utility functions found here: a2Stack.c
- **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
main and support functions	25
data types	20
instrumenting sum functions	20
instrumentation functions	35

- **Assignment grade:**
  - Your grade will be computed based on two criteria:
    - *completeness* of the program functionality and its execution
    - *quality* of the implementation
  - You must familiarize yourself with the grading rules
- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features