Helwan University
Faculty of Computers and Information

Sheet2

Department of Computer Science
Data Structures
Spring 2007

This sheet is an educational one that aims at teaching the student the concept of modular programming and how to work with several modules (files) at the same time. The number of files in this very-small project is just 3. After getting familiar with the concept we will work with more files in the future, as applications require. **Code all of the problems before attending the lab**.

1. Write the implementation level of the array-based stack in a separate file and name it `stack.c`. Compose the header file `stack.h` that consists of the prototypes of the functions along with the definition of the stack element type `StackEntry`. For the moment, we can define `StackEntry` to be a character. Now, you have to include `stack.h` in `stack.c`; why?
2. Compile `stack.c` and make sure that the compiler generated the object code file `stack.obj`.
3. Write a test program, name it `main.c`, as explained below, that calls the stack functions. Of course, you should include the stack header file here; why? The program should start, automatically, with creating a stack; why? Then, the program should display the following menu:
   (a) Clear the stack.
   (b) Read an element then Push it.
   (c) Pop an element then display it.
   (d) What is the stack size?
   (e) Exit.
4. Try to compile—not build—`main.c`. It compiles because the input-output parameters of the functions, along with the type `StackEntry`, are defined in the header file `stack.h`.
5. Build your project; this is the step in which the "Linker" links the functions from `stack.obj` to `main.obj` to create one executable file `main.exe`. Run your program.
6. In `stack.c`, write the function `StackTop` the returns a copy of the top-element of the stack without deleting it. Add this option to the menu of the main program.
7. Assume that you purchased the stack library `stack.obj`, along with its header `stack.h`, from the author; i.e., **you do not know any of the details of** `stack.c`. Rewrite the previous function in the application level `main.c`.
8. Assume that the function `StackSize` is not provided in the implementation level, but the other functions are. Write the function in the application level and test it in the program.

Hints: Every function in the implementation level assumes some pre-conditions. For example, the function `Pop` assumes that the stack is not empty. So, it is the responsibility of the user, at the application level `main.c`, to check for the stack status before performing any `Pop` operation. Similar pre-conditions apply to other functions; make sure that you do not overlook these pre-conditions.