Assignments P2.1 - Part 3

General remarks

All assignments are to be programmed in *portable* Fortran 2003/2008. **In addition**, all programs have to be “valgrind clean”, i.e. valgrind’s memcheck tool should report: *All heap blocks were freed -- no leaks are possible* and also: *ERROR SUMMARY: 0 errors from 0 contexts* unless the reported errors are caused by the compiler (e.g. Intel Fortran). The assignments in this part depend on completion of *all* sections from part 1.

16 Find next pow2

Write a function *next_pow2(n)* that returns the next integer that is larger or equal than *n* and a power of two.

17 Linked list

Integrate the provided file *17_array_lookup.f90* file into your build system from parts 1 and 2. *17_array_lookup.f90* provides a framework for testing and benchmarking data structures and looking up the previously stored data in random order. The file expects to read d3_#.dat files with data pairs. You need to implement operations 1) to initialize the data structure, 2) to add items to the list, 3) to look up items by value (i.e. the “key” element of the pair type); this should be a function returning the pair, and 4) to deallocate the data structure completely.

18 Hash table

Now implement a hash table using linked lists as buckets. Use the *next_pow2(n)* function to determine the actual number of buckets based on a suggested minimum value. Implement a function *hashfunc(val, size)* that takes the value to be hashed and the (power of two) number of buckets as an argument. This function shall contain an interface to the provided C function *inthash(key, size)* and the corresponding `inthash.c` source file needs to be added to the support library sources. To get it correctly compiled, the CmakeLists.txt file also needs to be changed to enable C as programming language besides Fortran.
You may alternately choose to re-implement the hashing function in Fortran and skip interfacing the C function. It must produce identical hash values as the C version.
Inside the hash table functions you can call hashfunc() to compute a hash based on the (integer) key of any data pair and it shall return the index of the bucket the data shall be stored in. Experiment with multiple minimum numbers of buckets (50, 100, 200). Add this to the testing and benchmarking framework as in section 17 and implement the same 4 types of operations. Discuss the performance of looking up elements by value in the three data structures.

19 Stack class with array

Implement a stack class for integers using a dynamically allocated array as backing storage. Implement a default constructor and a copy constructor, a free() function, and push(), pop() and length() methods. Write a test program that instantiates one stack object, fills it with data, then instantiates a second via the copy constructor from the first, and adds more data to both. Empty both stacks in a while loop each. Finally free all allocated storage and end.

20 Stack class with linked list

Program another stack class with the same API as in section 19 and use a linked list as backing storage this time. Discuss benefits and disadvantages of these two kinds of stack classes.