COMP281 Marking Descriptors for Assignment 3

This assignment assesses all the following learning outcomes:

1. analyse and explain the use of memory resources within software applications, including memory usage on the stack during function calls and heap-based dynamic memory management;
2. use debugging tools to inspect memory usage, and to assist in the development of software;
3. develop applications within the C programming language, including use of command-line driven C development tools.
4. deal with underlying memory-based issues in using dynamic data-structures through the implementation and management of at least one familiar datastructure using the C programming language.

The grade awarded is based on a profile formed from the following categories. The overall grade awarded is guided by this profile but not necessarily a weighted or averaged grade.

Failure in the assignment will not be compensated for by higher marks in other components of the module.

- All problems are weighted equally.
- For each problem, you can earn
  - 10 points for “Functionality and Correctness” awarded for programs that correctly solve the problem for all test cases.
  - 8 points for “Programming style, use of comments, indentation and identifiers” awarded depending on the style, comments and efficiency of the solution
  - 2 points for the quality and depth of the accompanying report
- The final grade results from normalizing the earned points to a scale of 100.
- See separate “COMP 281 Detailed Marking Guidelines” for more details.

You need to write a valid C program that solves each of the given problems – it must read the input, as specified in the problem description then print the solution to the given problem for that input. Note that code is “correct” only if it correctly implements a solution to the problem stated in the assignment, not "if online judge accepts it". That is, even if OJ accepts your code, it could be wrong. Read the problems carefully.

Category	A++ to A	B	C	D	E+	E- to G
Correctness of code	All problems are (nearly) correctly solved for all test cases. No important functional errors.	At least 2 of the problems are (nearly) correctly solved for all test cases. There is clear evidence of functionality on at least some of the test cases for the other problems.	At least 2 of the problems are (nearly) correctly solved for all test cases. At least one other problems shows some functionality on at least some of the test cases.	At least 1 of the problems are (nearly) correctly solved for all test cases and the other problems show some functionality on at least some of the test cases.	At least 1 of the problems is (nearly) correctly solved for all test cases.	No problem is correctly solved for all test cases. Little to no functionality on any of the problems.
Style of Computer Code	The code is well-presented, clear, efficient, well commented including meaningful variable names.	The code is mostly well-presented, clear and efficient. There may be some minor inefficiency or lacking in clarity.	Most of the code is reasonably efficient but may be lacking in some clarity or comments	There is a lack of efficiency in most of the code, and/or most of the code is unclear with little commenting.	There is some efficient code and/or some reasonable attempt at making the code readable.	Little or no attempt has been made to make the code readable; solutions may be very inefficient.
Detail in Report	In depth report describing the algorithms used in the code as well as any extra features	Report describes all of the algorithms and features, but with a lack of depth or clarity; may also describe most of the algorithms clearly and in-depth.	Report describes most of the algorithms and features with a lack of depth or clarity.	Report describes some of the algorithms and features, but with a lack of depth or clarity.	Report contains some detail about at least one of the algorithms or features, but possibly without depth or clarity.	No report or lacking any meaningful and understandable details.