Question 3 Part 'e':

Comparison of the four implementations from the perspectives of software engineering metrics:

Efficiency:

1. **Part a:** Perhaps the best feature of the code in C is the feature of efficiency. C programs are generally faster than C++ programs, due to simpler features, and being a lower level language than C++, they are closer to machine language, thus executed faster.

2. **Part b:** Though slower than C, this implementation is the fastest among all those implemented in C++. As separate stacks are defined for characters and integers, some level of initialization and type casting is saved, thus leading to faster execution.

3. **Part c:** This implementation is even slower than part 'b'. This is because a template is employed for implementing integer and character stacks, which has to be initialized to the proper type prior to use, leading to some costs in terms of computation.

4. **Part d:** This is the slowest among all implementations. Standard template libraries are inherently slower (due to the high level of complexity involved), leading to slower execution rate for the program.

Ease of implementation:

1. **Part a:** This factor is weak for this part, as C codes are not generally implementation friendly for complex types. The lack of classes in C is detrimental towards easy implementation practices for C. Lack of understandability of the code is observed in this part due to the weaker semantics of C.

2. **Part b:** Implementation is much easier in this part (implemented in C++) as compared to part 'a'. However, different classes have to be defined for the integer stack and character stack, which leads to a lengthier and cumbersome code.

3. **Part c:** This part is easier to implement than part b, as we can do away with the different stack implementations for integer stacks and character stacks by the means of template feature. However, we still have to define the functions and members for the stack class.

4. **Part d:** This is by far one of the best advantages of using the Standard Template Library (STL). Pre-defined functions in the library 'stack' dramatically reduces effort from the

user for stack implementation. This leads to reliable, safe and semantically much neater code implementation.

Testability:

1. **Part a:** Testability is not a strong feature for C codes such as part ‘a’ due to several reasons: code stability is weaker in C due to weaker semantics; exception handling is not as efficient in C as in C++; and input – output methods in C are not as versatile as in C++.

2. **Part b:** Testability is much better in C++ as compared to C, by the reasons stated in part ‘a’. However, as much of the code has been implemented by the programmer, the program is more open to exceptions and prone to testing vulnerabilities.

3. **Part c:** Testability is better in this part as compared to part ‘b’, because a common implementation is present for both the character and integer stacks, thus reducing the scope of exceptions. However, still, there is some level of programmer implementation of the stack, which still leaves scope for vulnerabilities while testing.

4. **Part d:** Testability is much better for this part. Reliable standard template library ‘stack’ leaves much less scope for exceptions to occur, giving rise to better testability options due to simpler and more understandable implementation. Tests of more complex nature can be thus carried out in this part.

Robustness and maintainability:

1. **Part a:** This part implemented in C is an instance of Ward Cunningham's concept of technical debt, which is an expression of the costs resulting of a lack of maintainability. Difficult - to - read code and poorer semantics characteristics of C greatly reduce the robustness and maintainability of part ‘a’.

2. **Part b:** Greater robustness and maintainability is introduced in this part implemented in C++ as compared to part ‘a’. But a greater amount of code for the character and integer stack leads to lower proportions of maintainability for the maintainers.

3. **Part c:** Reduced code length by the virtue of template implementation increases the scope for maintainability of the code, adding to its robustness. Still, the code for all the stack functions must be evaluated and maintained.

4. **Part d:** The robustness and maintainability of this part is much better than all the others, due to dramatic code reduction with the help of the standard template library ‘stack’.

Reliable implementation of stack in the library leaves for the maintainer only the maintenance of the code for the use the functions and features in this library.

Readability:

1. **Part a:** Poor coding semantics in C render the readability of part ‘a’ a difficult task for the programmer. Repetition of bulky code, lack of object oriented programming, etc. reduce the ease for a programmer to read and understand the code.

2. **Part b:** Object oriented programming increases the readability and conceptual clarity of the code for the programmer. However, repetition of code for separate implementations of character and integer stacks increases the bulk of the code, thereby reducing readability.

3. **Part c:** The use of template removes code repetition for stack implementation for character and integer stacks, thus improving readability. Still, some level of programmer implementation remains, which must be read and evaluated by any second programmer.

4. **Part d:** Dramatic code reduction by the use of standard template library ‘stack’ greatly enhances readability and understandability for a second programmer evaluating the code. Direct function use from the library facilitates the programmer to understand the algorithm, rather than spending too much time over the code semantics.

- Siddharth Rakesh 11CS30036