

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | |
|  | | | High Integrity Software DevelopmentCOURSEWORK REPORT | | | | | | | | | |  | | | |
|  | |  | | | | | | | | | | | |  | | |
|  | | | | | |  | | |  | | | | | | | |
|  | | | | | | ALEXANDROS **RANTOS**  **CHARISOPOULOS**  Student Number  **190587101** | | |  | | | | | | | |
|  | | | | | | DATE 12/12/2019 | | |  | | | | | | | |
|  | | | | |  | | | | |  | | | | | | |
|  | | | | TASK 1: CODE ANALYSIS | | | | | | |  | | | | | | |
|  | | | |  | | |  |  | | |
|  | INFORMATION FLOW In the specification file in order to declare the dependency between the procedure’s variables, depend clauses were used. More specifically, in the first procedure of the exercises.ads file (Calc) variable Result can depend on the Cost variable hence adding is mandatory. Similarly, in the next procedure Quantity value is multiplies by a constant and is assigned to Charge so thishad to be added. For the rest procedures, the same pattern follows but since both write on an array it must be clarified that the array return value is depending on the index of the array and the already content of itself. This is done by the following lines & on the Modify & MarkSequence procedures correspondingly. | | | | | | | | | | | | | | |  | |
|  | CORRECTING RUN-TIME ERRORS Many run-time errors appeared when tried to run the proof tools of SPARK for the given project. As it can be seen in the picture attached below, these errors had to do mostly with overflows, array indexes and unititialized variables.    In order to fix the second error in the list of the picture, Charge local variable had to be initialized in the .adb file so the declaration of the variable was changed to this line : ADDITION OF PROOF ASPECTS Pre & Post aspects should be introduced not only to restrict the input & output values of our procedures and to verify their results. In detail the following steps were taken in .ads file for each procedure:   * Calc : Since we multiply Cost with 100 we must check if the result does not overflow so we add this Pre statement. As far as the result of this procedure a Post aspect had to be added to eliminate the possibility of assigning an undefying value to the Result variable this adding this line prevented this . * CalculateCharge: As explained above for the same reasons Pre and Post aspects had to be introduced to tackle similar overflow and unknown value errors. * Modify: The primary goal was to find the right Pre condition that could fit every possible combination of B + C that wouldn’t overflow the system and still be in the range of Size. For that specific task this complicated pre aspect was written Once again it is checked that the final value is indeed correct * MaskSequence: For this one the proof tool had to deal with checking every single array value so for all keyword was used. In the Pre aspect, both arrays length are checked if they are within the range of Size and then every element of Target Array is checked if it is less than the maximum integer value and then if all values of Mask array are either 1 or 0.     For the Post condition the final values of the Target array are verified that have been assigned the desired value.    In conjunction with the above conditions the following loop invariant had to be added so in every loop it is certain that the values are correct up to the index of that loop. | | | | | | | | | | | | | | |  | |
|  |  | | | | | | | | | | |  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | TASK 2: WEAKEST PRECONDITIONS | | |  |
|  |  |  |  |  |
|  | PROOF OF VERIFICATION   The Hoare Triple is {true} A := 2\*B; Result := A\*A {Result < 144}  And the verification condition is P -> wp (S,R)  {true} -> wp(A := 2\*B; Result := A\*A, Result <144) [Hoist PostCondition]  {true} -> A\*A < 144 [Hoist using A := 2 \* B]  {true} -> (2\*B) \* (2\*B) < 144  {true} -> 4 \* B2 < 144  {true} -> B < 6 : vc , H : true & C : B <6 which is not provable.    If-Then Rule is applied: {P} if S then C end if {Q}  Formally:  H1: {P and S} C {Q} , H2: (P and not Q) -> Q  C : {P} if S then C end if {Q}  So H1 is:  {M > N and M > N + 10} M := M – 10 {M > N}  {M > N and M > N + 10} -> wp( M := M – 10, M > N)  {M > N and M > N + 10} -> M > N [Hoist PostCond]  {M > N and M > N + 10} -> M – 10 > N  {M > N and M > N + 10} -> M > N + 10 ,  so wp( M := M – 10, M > N) = M > N + 10   * 1. {M > N and M > N + 10} -> M > N + 10 (1st vc),   H1 C1  Provable via and elimination   * 1. {M > N and not M > N + 10} -> M > N + 10 (2nd vc),   H2 C2  Provable  Since H1 & H2 are both provable we can assume C. | | | | |  |