My C++ program (BubbleSorting.cpp) prompts the user for five test scores as integers greater than or equal to 0 and less than or equal to 100. It ensures that the user entires are valid and stores them in an array. Then it uses the bubble sort to sort the scores in ascending order and calculates the letter grade for each test. It also calculates the average score and displays the results. I used bubble sort method in lines 161-178. First, I passed testScore array as a variable into bubbleSort function. For example, if the user inputs an array of 65, 43, 76, 89 and 93, then those five tests will first be validated to see if they're integers and go to bubbleSort function. In line 165, I used a for statement "for(int = 0; i < 5; i++)," so that it checks every single element in that array from 0th one to the 4th one (5 elements). Then I used another for statement similar to the previous one, and inside that for statement, I created an if statement "if(testScore[j-1] > testScore[j])" to swap numbers to display them in ascending order. For instance, if the first element is 65 and the second one is 43 in testScore array, then that if statement checks what's greater. In this case, testScore[j-1] would be testScore[0]=65 and testScore[1] = 43, and 65 is greater than 43. Since it satisfies the condition, the program will go through this loop. Once it's inside the if loop, 65 is saved in temp to be compared with the next element, 76 later on. This procedure keeps repeating until all elements are in the right place. This is why bubble sort is inefficient for an array with large number of elements because it literally has to compare each element with the adjacent one.

My Java program (BinaryToBaseTen.java) uses simple mathematical algorithm to convert binary numbers to their base 10 equivalents. When the user enters a binary number, it first validates if it really is a binary number. Then it gets passed into the binaryToBaseTen function as a variable. Below code in lines 47 to 52, it converts a binary to decimal number.

47 while (binaryNumber != 0)  
48 {  
49 base += (binaryNumber % 10) \* Math.pow(2, power);   
50 binaryNumber = binaryNumber / 10;  
51 power ++;   
52 }

If the user inputs 111, it's going to go through this while loop three times like this,

|  |  |  |  |
| --- | --- | --- | --- |
| 47 | 111 != 0 | 11 != 0 | 1 != 0 |
| 48 | { | { | { |
| 49 | base += (111%10) \* 2^0 = 1 | 1 += (11%10) \* 2^1 = 2 | 3 += (1 % 10) \* 2^2 =4 |
| 50 | binaryNumber = 111/10 = 11 | binaryNumber = 11/10 = 1 | binaryNumber 1/10 = 0 |
| 51 | power ++ = 1 | power ++ = 2 | power ++ = 3 |
| 52 | } | } | } |

After the third time, binaryNumber becomes 0, so it gets out of the while loop and returns to base which is 1 + 2 + 4 = 7. It is correct because 111's base 10 equivalent is 7. I had to use binaryNumber % 10 and binaryNumber / 10 to get each 1 and 0 in entered binary number and to multiply that number by 2 to the n-th power.