1) The uninitialized variable is causing an error in valgrind. To fix this, I simply initialized the variable to zero. In order to free the definitely lost memory, inside the for loop that it was defined in, I freed the pointer to definitely lost first and then the definitely lost variables. The first pointer to definitely lost freed up the indirectly lost issue, and the second freeing of definitely lost freed up the definitely lost memory. The still reachable variable was just that, still reachable. It needed a simple freeing of the variable and that cleared it in valgrind. The possible lost was allocated like the still reachable, but adding 4 to the pointer caused the pointer to move forward four and to point at junk. To free the original memory, I subtracted the four and then freed the possibly lost memory. This resulted in all bytes of memory being freed!

2 ) This program had a few memory leaks and mishaps that needed to be fixed so that it could run the way it was written to. To accomplish this, I used Valgrind, a memory leak checker, and GDB, a debugger for C and C++, as well as the materials provided for the lab. The first error came with the number of bugs on Mars. I used Valgrind to figure out that this variable was initialized after the call to print the variable, which is why it was not working. I moved the initialization to occur before the print, which fixed the error. The next error was discovered by using GDB and found to be with the echo function, which prints until it hits a null. The sentence structure did not have this at the end, so it needed to be added. It was discovered that this was a buffer overflow error. The echo was going down the structure, printing "butterfly" (even though it wasn't supposed to), and then trying to print the huge number that was next, but it was expecting an array of characters, which caused the overflow and seg fault. The next error was found using GDB with breakpoints to discover a seg fault error occuring with the echoohce function. The first problem was that it was printing the "iter" variable that had not been dereferenced. The function was also printing a funky "spider" variable that it was not supposed to be printing. This came from a similar error as earlier, that there was not a "NULL" to halt the loop. To fix this, I used the stop_beginning variable as the cap on the for loop so it would halt after it hit the last word in the sentence and not the spider variable. I commented out the "free useless bug" and the code seemed to run just as it was supposed to. However, when I used valgrind to check that I didn't have any memory leaks, it happend I had 8 bytes that were definitely lost. I watched one of the lab posted videos about pointers, and discovered that this was occuring because the same variable was being allocated twice, so the original allocation had been lost. To fix this, before the second allocation, I made sure to free the variable, which fixed the problem. The current bug adjective was supposed to print as null, so after freeing the third element of the sentence, I set it equal to NULL.