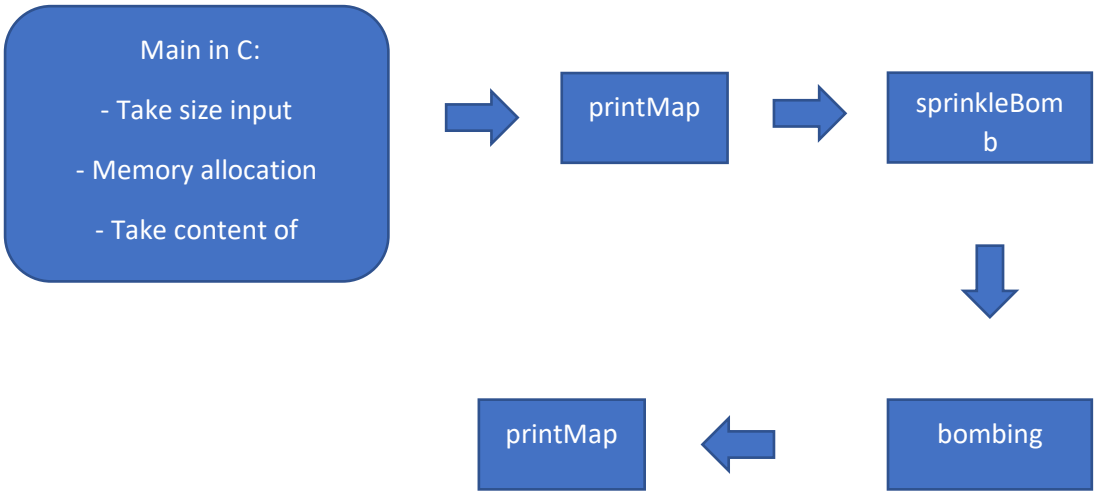
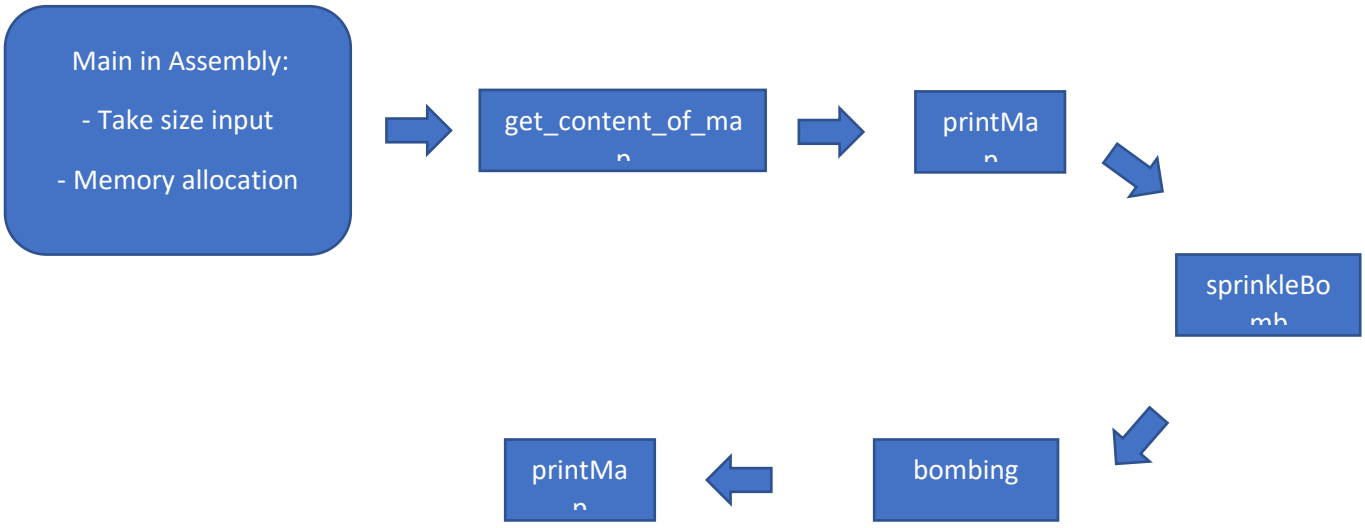
CSE 331 Computer Organization Homework-1 Report

The Assembly code I wrote take input from the user one by one (character by character). First of all, when I wrote the C code, I used a character array to hold the characters and an integer array to check whether the bombs were old bombs or new bombs. In C, I get the row and column values from the user and allocate memory in the main function. Then again in main, I get the contents of bombMap as a single string from the user. After getting the elements, I call the printMap function. This function prints the values entered by the user on the screen as a map. After that, I call the sprinkleBomb function, which sprinkles bombs in places other than the places where the user entered bombs and the entire map is covered with bombs. After doing this, the bombing function is called, the first bombs are detonated under the conditions requested from us and then the final output is printed on the screen again with printMap.





Some of the problems I faced and my solutions:

At first, when I tried to translate taking inputs as a single string in C to Assembly, I had some problems and I directly abandoned that method. I decided to take the characters one by one to make it easier for me. But this time I was also taking the enter (\n) character after the entered character and I could only take half of my inputs. Then I rewrote the line "li $v0, 12 syscall" after taking the input and thus I could take the \n characters and ignore them. This is how I solved this problem.

To check for old bombs in C, the first time the user put a bomb, I put 1 in the same index of the timeFlag array and 0 in other places. However, in Assembly, I set the timeFlag array as a character array in order to be more compatible with my code, easier and less memory consuming. This time I labeled the old bombs as 'x' and the new bombs as 'y'. In this way, when allocating memory, I allocated memory of equal size for both arrays and my work became easier later on.

When I first started the code, I wanted to keep the first address of my pointers in a global and fixed variable for convenience. I couldn't do it with '.space' without fix size. Then I created two 4 byte variables named bombMap and timeFlag with '.word'. Then, after allocating memory in the main function, I was assigning the first addresses of the allocated memory to these variables. I did this part in the first part of the assignment. Then I wrote all the functions, but I was getting ridiculous output that I didn't want. My bombing function, which I checked many times, was not working as desired. Then I started looking for problems in other functions, but I couldn't see any problems anywhere. In order to make better checks, I tried to print statements somewhere to make better checks, but I couldn't make sense of the outputs again. Then I thought of printing the contents of the timeFlag. I thought that at least that should work correctly, but I saw some very strange things in the output. The last 4 elements of the timeFlag array, no matter what size, were printed correctly, but the rest of the elements consisted of '.' and 'o' characters. I couldn't make any sense of this output because I had written expressions to input only 'x' and 'y' characters. To understand why there were '.' and 'o' characters there, I looked at what the contents of the bombMap array should be and realized that actually some of the contents of bombMap and the last 4 elements of timeFlag were being printed. After that I started to keep track of the address values stored in the registers and realized that for some reason there was always a 4 byte memory difference between my two arrays and when the code was fully compiled they both pointed to the same place. At this point I started to suspect the addresses where the arrays were stored and realized that the error was caused by the bombMap and timeFlag variables that I had set to '.word'. Since they were both 4 bytes, they were created one after the other and so the last 4 elements were found correctly. The elements printed before were the elements of bombMap. I solved this problem by deleting the variables I had set as '.word'. When the memory allocation was first done, I put the first addresses of the allocated memories in the $s6 and $s7 registers and edited the rest of the code accordingly. From that point on, my code started working flawlessly.