Names of Files to Submit: **matmult.s, knapsack.s, combs.s ReadMe.txt**

- If you are working in a group **ALL** members must submit the assignment
- All programs should compile with no warnings when compiled with the -Wall option
- All prompts for input and all output must match my prompts/output. We use a program to grade your work and tiny differences can cause your work to be marked as a 0.
    - The best way to avoid being deducted points is to copy the prompt/unchanging portion of the outputs into your code. Make sure to get the spaces as well.
- You must also submit a file called ReadMe.txt. Include the names of all partners and any trouble you had on the assignment
- An example ReadMe.txt has been included with this assignment
- The input in the examples has been underlined to help you figure out what is input and what is output
- Submit your work on Smartsite by uploading each file separately. Do not upload any folders or compressed files such as .rar, .tar, .targz, .zip etc.
- If you have any questions please post them to Piazza
- **RESTRICTIONS:**
    - For all programs in this homework you
        - May not have a data section. If you need extra space you must use the stack.
        - Not call any C functions that you write. You may make use of any of the library functions.

1. (Time: 3.5 hours. Lines of Code 257) Write a program called **matmult.s** that implements matrix multiplication in assembly. If you don't know how to do matrix multiplication your can find a tutorial here.
    1. This program should be callable from C and have the following signature:
        1. `int** matMult(int **a, int num_rows_a, int num_cols_a, int** b, int num_rows_b, int num_cols_b);`
        2. This function should multiply matrices a and b together and return the result
        3. You must allocate space for this new matrix by calling malloc
    2. You have been given a C file called main.c that accepts as command line arguments the names of two files that contain the matrices to be multiplied. main.c will read in these matrices, call your function, and then display the result. After it has displayed the result it will then free the space that has been malloced.
        1. Your function must be callable by this file
    3. You have also been given a makefile that should compile your program. Your program **MUST** be able to be compiled by this makefile. For those of you running 64 bit versions of Linux you may need to install the 32 bit binaries. I was able to do this on my machine by installing multilib for gcc. To do this I clicked on the Ubuntu Software App and then searched for multilib. I selected the one for gcc, installed it, and everything was good to go. If that doesn't work for you, you might find this this post helpful. If neither solution works for you please Google and see what you can find. If you find a solution that works for you please post it to Piazza.

        Example:

```
cat mata/0-test.txt
3
3
470 -192 -539
235 -814 -538
-503 -418 541

cat matb/0-test.txt
3
3
313 531 802
26 860 -767
543 870 822

./matmult.out mata/0-test.txt matb/0-test.txt
-150559 -384480 81146
-239743 -1043315 370572
125456 -155903 361902
```

2. (Time 1.5 hours) Write a program called **knapsack.s** that solves the 0-1 knapsack problem **recursively**. In the knapsack problem you have a knapsack that can hold W weight. You also have a collection of items that each have their own weight w_i and value v_i. The goal is find the set of items that maximizes the amount of value in the knapsack but whose weight does not exceed W.
    1. This program should be callable from C and have the following signature
        1. `unsigned int knapsack(int* weights, unsigned int* values, unsigned int num_items, int capacity, unsigned int cur_value)`
        2. This function should calculate and return the maximum value knapsack
        3. This function must be implemented **recursively**
        4. Pay very careful attention to the **types** in this function as it will affect which machine instructions you should use. Hint: it will affect the jump instructions you use
        5. You have been provided with a C file called knapsack.c that implements this function and should give you a good starting point
            1. If you want an extra challenge try solving the problem without looking at knapsack.c as this problem boils down to just finding the optimal *combination* of items
        6. You will find the `leal` instruction very helpful for this problem
    2. You have also been given a file called main.c that will take as a command line argument the name of a file containing a knapsack problem.
        1. Please see the comments in main.c to see how these files are structured
        2. Your function must be callable from this file
    3. You have also been given a makefile that will compile the assignment for you as well as create a version of the executable using only the given C files.
        1. Your program must be able to be complied by this makefile

    4. Example:
        1.
```
cat Tests/0-test.txt
100
4
43 43
3 38
5 17
18 25

./knapsack.out Tests/0-test.txt
123
```

3. (Time 1.5 Hours) Write a program called **combs.s** that generates all the possible combinations of a set of items of a given size.
    1. Your program should be callable from C and have the following signature
        1. `int** get_combs(int* items, int k, int len)`
    2. This function should generate all possible combinations of items taken k at a time and return a 2-D array where each row contains one combination
        1. The combinations should be added to the 2-D array in their natural order
        2. This 2-D array should be dynamically allocated
        3. As a hint you will probably need to develop a helper function that actually computes the combinations
    3. You have been given a file called main.c that will get the inputs and call your function
        1. Your function must be callable from this file
        2. You will also find some helpful functions in main.c that you can call from your program
    4. You have also been given a makefile to compile your program.
        1. Your program must be compilable by this makefile.
4. Examples
    1. ```
       How many items do you have: 5
       Enter your items: 1 2 3 4 5
       Enter k: 3
       1 2 3
       1 2 4
       1 2 5
       1 3 4
       1 3 5
       1 4 5
       2 3 4
       2 3 5
       2 4 5
       3 4 5
       ```
    2. ```
       How many items do you have: 5
       Enter your items: 1 2 3 4 5
       Enter k: 4
       1 2 3 4
       1 2 3 5
       1 2 4 5
       1 3 4 5
       2 3 4 5
       ```