COP3502C Computer Science 1
Dr. Andrew Steinberg
Spring 2022

Programming Assignment 1

Tell me a Random Story

Max Points 100

Due: 2/6/2022 at 11:59pm

Background Story of this Assignment

All of you should of passed COP3223C Introduction to Programming in C (or a similar course) to be enrolled in CS1. Some of you took the prerequisite last semester, some of you may have taken the prerequisite two semesters ago, and some of you may have taken the prerequisite over a year ago or more. Nevertheless, the goal of this assignment is to serve as a warmup with dynamic memory as it will play a heavy role in learning our data structures for CS1. If it has been months since you worked with C, then you may feel a little rusty (don't worry that is normal). That is why the first assignment is a review from some prerequisite topics (such as strings and dynamic memory). However, it is critical, to regain these skills in dynamic memory, strings, and structs (even though this assignment doesn't cover structs) so we can smoothly cover data structures and learn their ADT.

In this assignment, we will touch upon the topics of strings and dynamic memory. You are going to create a program that generates random sentences. These random sentences won't make sense however we will still follow the general rules of English of forming one. Since we are going to generate random sentences, we will also create the most random story that won't even make sense.

Start Early and see the TAs and ULAs! They are here to help you! Don't procrastinate!

Assignment Details

This section covers the assignment in more detail.

- The random sentence generated will be formulated by these types of words.

  $$\text{sentence} = \text{article}_1\ \text{adjective}_1\ \text{noun}_1\ \text{verb preposition}\ \text{article}_2\ \text{adjective}_2\ \text{noun}_2$$

  Note: The subscript represents that a different word of the respective category can be generated. For example $\text{article}_1 = \text{"The"}$ and $\text{article}_2 = \text{"A"}$.

- Here is a sample random sentence generated from the working program using the sentence formula from above. Sometimes words picked from respective categories can be the same (such as the article and adjective in this case). Note that there is one blank white space between each word and a period at the end of the sentence.

  ```
  An big year use against An big case.
  ```

  Here is another example where each word from the categories are all unique.

  ```
  A new group think into The early day.
  ```

- You might notice that the first letter in articles are all capitalized. This is due that the sentence formula starts with an article. Make sure for the articles, that they are all capitalized with the first letter. It should already be provided that way for you in the C file on Webcourses. Do not change it!

- You will use 5 separate dynamic 2D arrays (utilizing double pointers) in storing the respective words in their respective categories (noun, verb, adjective, preposition, and article).

The Provided Skeleton File

You were provided with a skeleton C file that has the main function and function prototypes. This section will discuss the lines of code provided for you in the main function to assist you with understanding how the code will execute.

- Lines 6 – 9 are preprocessor directive statements. The first three import content from the standard I/O library, standard library, and string library. This will allow you to use certain built in functions in completing the assignment.

- Line 9 defines a macro constant called LIMIT that holds the value 20. This will be number of characters a string can store. That means the words being read can only be of size 19 (we have to include \0 character).

- Lines 11 – 16 are the function prototypes. This is discussed in the next section in more detail.

- Lines 18 – 126 is the main function. Let's divide up parts of the main function.
    - Lines 21 – 24 sets up the randomness of our program. This will low the script the grader use in determining a random scenario to truly check your work properly.
    - Line 24 calls `srand()`. This is a function part of `stdlib` that generates a seed for the pseudo random generator. This will allow you to run the code with the same scenario repeatedly. If you change the seed, then you will get a different scenario which will result in a different output.
    - Lines 27 – 30 declares/initializes variables that keeps track of the size each dynamic 2D arrays of each category respectively currently stores. In simple terms, it will tell you how many words are in each category. They start empty which is why is 0 is the first value assigned.
    - Line 32 – 36 declares/initializes variables that keeps track of the max size each dynamic array can hold. In simple terms, it keeps track of how much the heaps can store fully (number of elements). Note: The number of articles used is 3 ("a", "an", and "the"). That is why `const` is used since we will not be changing the size of the array during the program run.
    - Lines 41 – 45 declares 5 double `char` pointers and assigns them to `NULL`.
    - Lines 47 – 51 assigns each of the double `char` pointers to a heap of single `char` pointers.
    - Lines 54 – 59 checks to make sure that the previous `malloc` call was successful in providing an address to a heap. If it was not successful then, the program terminates.
    - Lines 62 – 67 populates the articles 2D array with the respective three articles used in the English Language. It also allocates a heap to store the string.

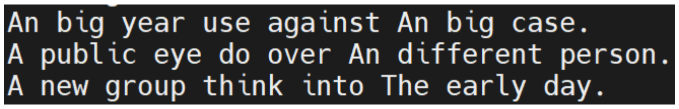
- Lines 70 – 73 opens the text files of words of each respective category. Those text files must be in the same directory as your C file.
- Lines 76 – 81 checks to make sure the files were properly opened, otherwise the program terminates.
- Lines 84 – 87 calls a user defined function (you will have to implement) `populate` that will fill the dynamic arrays with the respective words in their category. Details of `populate` are in the section.
- Lines 90 – 93 closes the text files since they will no longer be needed. The words should be in the 2D dynamic arrays.
- Lines 99 – 104 contains a for loop that will generate some random sentences. This is done by the user defined function `generatesentence` (which you will implement). Line 101 specifically calls a user defined function `displaysentence` that will display a random sentence to the terminal window. Line 103 frees the heap to prevent memory leaks. Think about how this will relate to the function `generatesentence`. That is part of the challenge.
- Lines 106 – 117 will open 3 text files to store 3 random stories respectively.
- Line 119 calls a user defined function `cleanUp` that frees any heap memory in use. You will have to implement this user defined function. That is part of the challenge.

The Function Prototypes

You are going to implement 6 user defined functions. In the provided main function, you were provided the prototypes. This section will describe what it is expected from each definition.

```
void displaysentence(char * sentence);
```

The `displaysentence` function will display the sentence to the terminal window. The function takes one argument which is a `char` pointer (dynamic string) and will display it on a single line. Here is a sample screenshot of the function being called three times consecutively.



```
An big year use against An big case.
A public eye do over An different person.
A new group think into The early day.
```

```
char ** populate(char ** words, FILE *fptr, int *currentsize, int *maxsize);
```

The `populate` function will assign actual values to the dynamic array. It will fill it with strings from the provided text files. The function will return the populated 2D dynamic string array. The functions takes 4 arguments:

1. A double pointer of type char called `words`. This is a 2D dynamic array that contains words (strings) from the respective category (noun, verb, adjective, preposition).
2. A file pointer that contains the location of the text file from being read. The file should be open in the main function.
3. A pointer of type int called `currentsize`. This pointer holds the address of an integer that keeps track of the amount of strings stored in the 2D dynamic array.

4. A pointer of type int called `maxsize`. This pointer holds the address of an integer that keeps track of the maximum amount of strings the 2D dynamic array can store.

There are some things you need to consider for this function implementation.

- When allocating heap space for the string, make sure it was successful. Remember `malloc` doesn't always return an address. Perform the necessary action recommended if that is the case. It is good practice to prevent segmentation faults.
- When the program starts running, the 2D dynamic array starts with 5 strings it can hold potentially. However, you will notice that the provided text files has more than 5 strings to read. That means we will need to allocate more heap space some how. Think about how `doubleIt` will work in this scenario. That is part of the challenge in this assignment.

```
char ** doubleIt(char **arr, int *maxsize);
```

The `doubleIt` function will increase the heap size by doubling the original space. The functions takes 2 arguments:

1. A double pointer of type char called `arr`. This is a 2D dynamic array that contains words (strings) from the respective category (noun, verb, adjective, preposition). This is the original heap.
2. A pointer of type int called `maxsize`. This pointer holds the address of an integer that keeps track of the maximum amount of strings the 2D dynamic array can store. When this function is invoked, the current size of the dynamic array is at capacity so the values should match. This value will need to be multiplied by 2.

The function allocates a new heap that is twice the size of the original heap and copy the values from the original heap into the new heap. One important note about this is that you have to properly copy the values over in order to avoid segmentation fault. Think about how this can be avoided. That is part of the challenge of this assignment. Avoid using `realloc`, `calloc`, or any sort of other memory function except for `malloc` and `free`. If any other built in memory function is used then the ones stated will result in points being deducted!

```
char * generateSentence(char ** noun, char ** verb, char ** adjective, char
** preposition, char ** article, int nounsize, int verbsize, int adjsize, int
prepositionsize, int articlesize);
```

The `generateSentence` function will create a string by combining the respective word categories into one string. The functions takes 10 arguments:

1. A double pointer of type char called `noun`. This is a 2D dynamic array that contains words (strings) from the respective category noun.
2. A double pointer of type char called `verb`. This is a 2D dynamic array that contains words (strings) from the respective category verb.
3. A double pointer of type char called `adjective`. This is a 2D dynamic array that contains words (strings) from the respective category adjective.

4. A double pointer of type char called `preposition`. This is a 2D dynamic array that contains words (strings) from the respective category preposition.
5. A double pointer of type char called `article`. This is a 2D dynamic array that contains words (strings) from the respective category article.
6. An int called `nounsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of nouns.
7. An int called `verbsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of verbs.
8. An int called `adjsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of adjectives.
9. An int called `prepositionsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of prepositions.
10. An int called `articlesize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of articles.

The function will create a dynamic character array that will store all components of the sentence into one string. The length of the string sentence will be no more than 100 characters (this includes the null character). After creating the new string sentence, the function will return it (as indicated by the prototype).

There are some things you need to consider for this function implementation.

- When allocating heap space for the string, make sure it was successful. Remember `malloc` doesn't always return an address. Perform the necessary action recommended if that is the case. It is good practice to prevent segmentation faults.
- Make sure the sentence has a space (one blank white space `' '`) between each word.
- Make sure to include the period symbol at the end of the sentence along with a newline character.

```
void generateStory(char ** noun, char ** verb, char ** adjective, char **
preposition, char ** article, int nounsize, int verbsize, int adjsize, int
prepositionsize, int articlesize, FILE *fptr);
```

The `generateStory` function will create a random story by putting a bunch sentences together and storing it in a text file. The functions takes 11 arguments:

1. A double pointer of type char called `noun`. This is a 2D dynamic array that contains words (strings) from the respective category noun.
2. A double pointer of type char called `verb`. This is a 2D dynamic array that contains words (strings) from the respective category verb.
3. A double pointer of type char called `adjective`. This is a 2D dynamic array that contains words (strings) from the respective category adjective.
4. A double pointer of type char called `preposition`. This is a 2D dynamic array that contains words (strings) from the respective category preposition.
5. A double pointer of type char called `article`. This is a 2D dynamic array that contains words (strings) from the respective category article.

6. An int called `nounsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of nouns.
7. An int called `verbsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of verbs.
8. An int called `adjsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of adjectives.
9. An int called `prepositionsize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of prepositions.
10. An int called `articlesize`. This variable holds an integer that keeps track of the amount of strings currently stored in the 2D dynamic array of articles.
11. A file pointer that contains the location of the text file from being read. The file should be open in the main function.

The function will create between 100-500 (both inclusive) sentences (use the `generateSentence`) to generate a random story (it won't even make sense) that will be stored in a text file. You will notice three story text files created that will store each story. Check out the examples posted in Webcourses. Based on the seed being utilized, those are the stories generated. One important aspect to consider for this function are memory leaks. Make sure there are no leaks!

```
void cleanUp(char ** nouns, char ** verbs, char ** adjectives, char **
prepositions, char ** articles, int nounsize, int verbsize, int adjsize, int
prepositionsize, int articlesize);
```

The `cleanUp` function will make sure to free memory from the heap being used before the program terminates. The function takes all of the 2D dynamic strings and their respective sizes that are used in the program.

Requirements

Your program must follow these requirements.

- The output must match exactly (this includes case sensitivity, white space, and even new lines). Any differences in the output will cause the grader script to say the output is not correct. Test with the script provided in order to receive potential full credit. Points will be deducted!
- Do not modify the function prototypes or else points will be deducted.
- Do not add/remove any user-defined functions.
- Do not use static arrays at all! Use dynamic allocation (`malloc`)! We will test your program with different input of different size. So make sure your arrays are flexible in dynamic allocation.
- You may NOT use `realloc`, `calloc`, or any sort of other memory function except for `malloc` and `free`. If any other built in memory function is used then the ones stated will result in points being deducted!
- Make sure you do not have memory leaks! Those are bad to have. You can check for memory leaks using valgrind. In the provided file, you saw `malloc` and `free` being utilized, however they are currently not correlated. Think about how they would be used in the respective user defined functions. You will have to call them in your user defined functions. That is part of the challenge to think about with dynamic memory. You will have to fill that in your user defined functions.

- Do not change ANY content of the skeleton that was provided for you. Your code will be tested through a script that relies on this main function flow along with the header file being imported. Any changes to this will result in your program not working fully which will lead to point deductions that will not be fixed!
- Do not change the name of the text files provided for you. Use the original names that were provided for you.
- The text files must be in the same directory as your C file.
- Name your C file `program1_lastname_firstname.c` where `lastname` and `firstname` is your last and first name respectively. Please make sure it matches the spelling exactly how it is registered in Webcourses. Points will be deducted if the file is not named correctly.
- Make sure you test that `malloc` returns an actual heap address as discussed in lecture! If `malloc` isn't successful, then terminate the program using `exit(1)`. There is a sample shown the main function from the provided C file. Make sure this statement doesn't get executed. Otherwise fix your code.
- Make sure you implement this in the C language. Using any other language will result in a score of 0 on the assignment.
- Make sure to follow good coding style and comment your code!
- Your code must work on Eustis. If it does not work on Eustis, points will be deducted and not changed as mentioned previously.

Tips in Being Successful

Here are some tips and tricks that will help you with this assignment and make the experience enjoyable.

- Do not try to write out all the code and build it at the end to find syntax errors. For each new line of code written (my rule of thumb is 2-3 lines), build it to see if it compiles successfully. **It will go a long way!**
- After any successful build, run the code to see what happens and what current state you are at with the program writing so you know what to do next! If the program performs what you expected, you can then move onto the next step of the code writing. If you try to write everything at once and build it successfully to find out it doesn't work properly, you will get frustrated trying find out the logical error in your code! **Remember, logical errors are the hardest to fix and identify in a program!**
- Start the assignment early! Do not wait last minute (the day of) to begin the assignment.
- Ask questions! It's ok to ask questions. If there are any clarifications needed, please ask TAs/ULAs and the Instructor! We are here to help!!! You can also utilize the discussion board on Webcourses to share a general question about the program as long as it doesn't violate the academic dishonesty policy.