**Documentation for**

**CPS1011 Programming Principles in C**

**Assignment 2020/2021**

Nathan Bonavia Zammit – 0198402L

In this word document I explained what each task that I completed in my project does, and where I struggled to complete said tasks, such as errors and so forth.

Question 1

a.)

i.) bool get\_text\_from\_user

In this function I prompted the user to insert a paragraph of text by using a printf() function. I then simply indicated the number of characters to be used using “buf\_size-1” and indicated that if the function were to work, it would return true.

ii.) bool clean\_and\_validate

This function is used to clean the data and make sure that the ascii characters used are printable characters. I did so by using an if statement to check that the last character is a new-line character and then if the if statement is true, it would replace the new-line character with a null character. Then to check that the characters were printable ascii characters I used a for loop to go through every single character, and then used an if statement to check if they were printable ascii characters or not (by checking that the values were between 36 and 126).

iii). void convert\_to\_lower\_case

This function is used to convert the inputted characters into lower case characters. To do so I used a for loop to go through every single character inserted, and I passed them through the tolower() function.

b.)

i.) int is\_word\_char

This function is used to check that the character that is passed is a word character, so basically letters, digits, hyphens, underscores, and apostrophes. To do so I passed the character through 3 different if statements. The first checks if it’s a letter using the isalpha() function, the second checks if it’s a digit using the isdigit() function, and the third checks if it is an apostrophe, hyphen, or an underscore by checking their respective ascii values. If any of these are true, it returns a 1, otherwise a 0 is returned.

ii.) int is\_sentence\_terminator

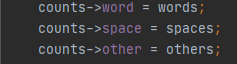
This function checks whether the character that is passed is either a sentence terminator, so a full stop, question mark, or an exclamation mark. To do so I used an if statement that simply checks if it is either of the aforementioned terminators, and if it’s true it returns a 1, otherwise it returns a 0.

iii.) int count\_sentences

This function returns how many sentences were in the supplied string. To do so I created a counter to increment whenever a sentence is detected, and 2 flags, named “current” and “previous”. I used a for loop to go through every single character in the supplied string, and used the is\_sentence\_terminator() function to check each character. If a sentence terminator is detected the count is incremented. However, a sentence could possibly end with multiple sentence terminators. Therefore, the flags are used to make sure that count is incremented only once. If a sentence terminator is indicated, current’s value is now 1, if the previous flag is equal to 0, only then is count incremented, otherwise nothing happens. At the end of the loop previous will end up with the value of current, so if both have their values set as 1, the count won’t increment. When a word character is indicated again, current’s value will end up 0 again, and so forth. The counter “count” is then returned holding the value of how many sentences are in the supplied string.

iv.) void get\_char\_counts

This function is used to count the word characters, space characters, and any other characters. I did so by first create counters for each character which I named “words”, “spaces”, “others”. I then used a for loop to go through each character and did if statements to check whether they were words (using the is\_word\_char() which I had done before), spaces (by checking if the ascii value was 32 since that is a space’s ascii value), or if they were other characters. When the respective conditions were met each counter was incremented respectively. For the “others” counter, it was incremented when neither of the conditions were met. These were then returned in the provided char\_counts\_t struct. “words” was returned to “word”, “spaces” was returned to “space”, and “others” was returned to other” as can be seen below:



v.) int count\_words

This function returns the number of words in a supplied string. To do so I did the same as I did in the count\_sentences() function, I created a counter and 2 flags named “current” and “previous”. I then used a for loop to check every single character in the supplied in the string. What I did differently however is that when a character is no longer a word character (is\_word\_char() returns 0), the counter named “count” is then incremented to indicate that the end of a word is reached. “count” is then returned, containing the number of words in the supplied string.

vi.) void get\_letter\_frequencies

This function counts the frequencies of the alphabetic letters in the text, and then stores them in the “freq” array. It then returns the maximum frequency which is provided in the “max\_freq” parameter. For this function I created a holder called “letter\_place”. Using a for loop I went to through every character that is passed and I checked them using the isalpha() function. It is then placed in the “freq” array. Then I created another holder called “max\_letter\_fequency”. I used a for loop and an if statement to compare the letter that is being passed’s frequency with other letter’s frequency. The one with the highest frequency is placed in the “max\_freq” parameter.

vii.) void get\_text\_statistics

In this function we had to use the previous functions that we made earlier to then populate the “statistics\_t” struct. The function get\_char\_counts() is used to obtain how many word characters, space characters, and any other characters. The function count\_sentences() was used to count how many sentences are in the supplied string, and then this was stored in the “statistics\_t” struct in “sentences”. The function count\_words() was used to count the words in the supplied string and this was then stored in the “statistics\_t” struct in “words”.



The function “get\_letter\_frequencies” was then called to obtain the frequencies of the alphabet letters in the supplied string as mentioned before. The frequencies obtained were then stored in the “statistics\_t” struct.

c.) bool get\_text\_file

This function is used to open a specified file and then read the characters (the paragraphs of text) found in said file, into the provided buffer of size buf\_size. To do so, I used the fopen() function to open the file and indicated that I wanted to read the contents of said file by indicating through “r”. This can be seen below:



I then did an if statement just in case the file was not to be found, i.e. it would be equal to NULL, which if true returns false. I then used a fscanf() function to go through the contents of the file with the help of a while loop, as can be seen below:

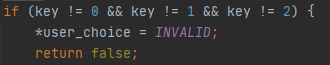


I then closed the file using the fclose() function and returned true in the end.

d.) bool print\_menu

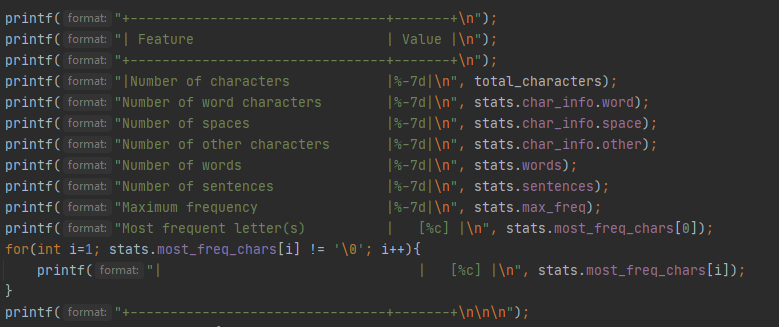
The use of this function is to present the user with a menu to decide whether to accept a paragraph of text via user input or file input, or to quit the menu.

To do so I printed a statement asking the user to choose a method of input or quit. Then I declared a key which is used to indicate which input is chosen. I then did an if statement just in case neither of the provided inputs are chosen to indicate that the user chose an invalid input, as can be seen below:



Then I used 3 if statements, one for each input chosen. If 1 was chosen then USER\_INPUT is stored in user\_choice, if 2 was chosen then FILE\_INPUT is stored in user\_choice, and if 0 is chosen then QUIT is stored in user\_choice.

e.) Main Method

In the “main\_1.c” executable code I set up 2 char variables, “buffer” to store the paragraph input, and “file\_name” to take the name of the file which is inputted from the user, and then I sent up 3 int variables. “letter\_frequency” was set up as an array to store the frequency of each letter, “max\_frequency” was used to store the highest letter frequency, “total\_characters” was used to hold the total amount of characters. I then set up a while loop to run the menu. The while loop will terminate when the user chooses “QUIT” because it would activate an if statement that changes the Boolean flag(“quit\_flag”) which is used for the while loop and sets it as true, which terminates the loop. In the while loop I called the function “print\_menu()” which I had previously declared. I then set up an if statement which is used to alternate between whether the user decides to input their own string or through a file. If the “user\_Choice” is equal to “USER\_INPUT” then the user is then asked to input some text. If an error were to occur then a message is displayed to the user. If the “user\_Choice” is equal to “FILE\_INPUT” then the user is asked to insert the name of the file. If an error were to occur in obtaining the file a message is displayed to the user. When both the text submitted by the user, or when the text from a file is obtained, the text is passed through the “clean\_and\_validate()” function which we had declared earlier in order to make sure the data is cleaned. The text now needs to go through the functions “get\_letter\_frequencies()” and “get\_text\_statistics()” however the text must be all in lowercase and as such the function “convert\_to\_lowercase()” is called and the text is passed through. After this “get\_letter\_frequencies()”” and “get\_text\_statistics()” are both called. The letter frequencies are stored in the array “letter\_frequency” and a for loop is used to go through every single letter in the supplied string. The total number of characters is stored in “total\_characters” as mentioned before. After all this, we were asked to print a menu which showcased: the number of characters, the number of word characters, the number of spaces, the number of other characters, the number of words, the number of sentences, and the most frequent characters in the supplied string. This was all done via the functions we had declared earlier. For the most frequent characters, a for loop is used just in case there were multiple frequent characters.

Question 2

a.)

i.) sparse\_set\_ptr ss\_init

With this we set up and initialised the sparse set data structure. I made sure that the capacity that can be stores is equal to or less than the maximum value + 1. Since this function did not fail, a pointer was returned, otherwise NULL would have been returned instead.

ii.) bool ss\_destroy

This function is used to clean the sparse\_set which is supplied. The free() function was used in order to deallocate the memory since it was previously allocated by a call to malloc and calloc in the previous function.

iii.) bool ss\_contained

This function is used to check whether an element is found in the sparse\_set or not. I used an if statement to check whether there even are elements in the sparse\_set, I then used a for loop to go through the dense array until the required element is found. If it is found then true is returned. Otherwise false is returned.

iv). bool ss\_add

The use of this function is to add an element to the sparse\_set. I used an if statement to check that the element is in the correct range of the sparse\_set (it is not less than 0 and not greater than the max\_value). If an element is not in the correct range a message is displayed to the user and false is returned. I then did an else statement and used the ss\_contained() function to check whether the element is already in the sparse\_set or not. If this is true, then true is returned. If the element is not in the sparse\_set, then it is added, and the number of elements is incremented.

v.) bool ss\_remove

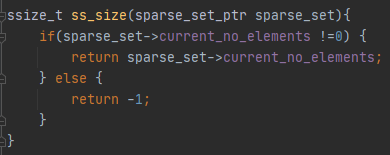
The use of this function is to remove an element from the sparse\_set. I used an if statement to check if the number of elements is not equal to zero, and then called the ss\_contained() function to check whether the element to be removes is in the sparse set or not. If both of these fail, then false is returned, otherwise the element is removed and true is returned.

vi.) bool ss\_sort

The use of this function is to sort the sparse\_set. I used an if statement to make sure that check that the number of elements is not equal to zero, otherwise false is returned. I then declared a temporary value. I used a nested for loop to go through all the elements and to compare the elements in the dense array. For example if dense[i] is bigger than dense[j] I swapped them using the temporary value, and did so accordingly with the sparse array aswell. If this were to succeed, true is returned.

vii.) ssize\_t ss\_size

The use of this function is to return the number of elements contained in the sparse set. I used an if statement to make sure that there are actually elements in the sparse set, if this condition isn’t met then -1 is returned, otherwise the number of elements is returned.



viii.) bool ss\_get\_elem

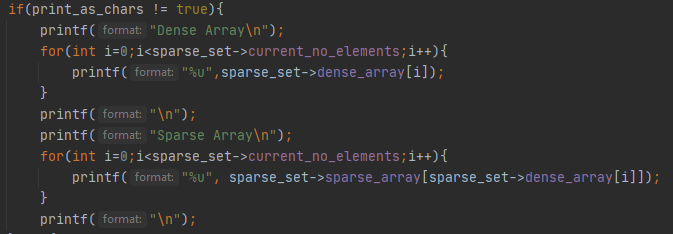
The use of this function is to return the element found at position sparse\_index at sparse\_set. The element as to be returned in dest. I simply placed the element inside \*dest by calling the sparse\_set and indicating which position I wanted to call:



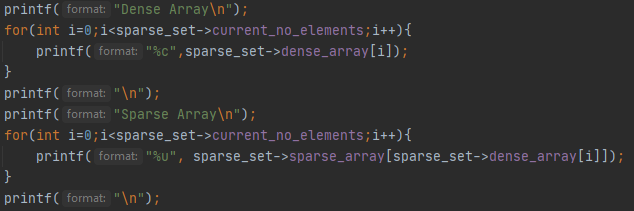
If this were to succeed then true is returned, otherwise if the value of dest ends up NULL, false is returned.

ix.) bool ss\_print\_status

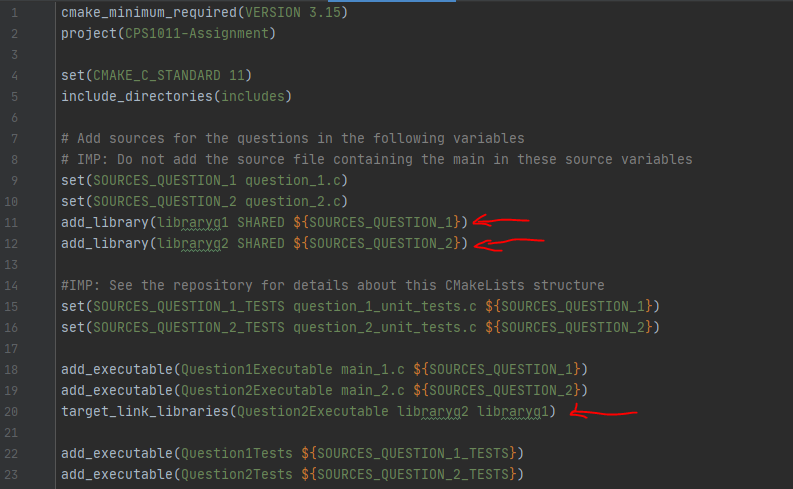
The use of this function is to print out the contents of the sparse\_set if print\_chars is true. Otherwise it’d be printed as unsigned integers. I used an if statement to check whether print\_chars is true or not. If it were false, then I proceeded to print the dense array and sparse array, using for loops to go through all the elements in the arrays, and used “%u” since they were to be printed as unsigned integers:



Otherwise, if print\_chars is true, then I proceeded to, again, print out the dense array and sparse array, using for loops to go through all the elements in the arrays, and used “%c” since they were to be printed as characters:



b.) The full implementation was compiled as a shared library in the CMakeLists file.



The lines 11,12 and 20 are indicated by a red arrow to show that they were added.

c.)

i.) bool ss\_union

The use of this function is to combine the elements in sparse a and b and place them inside result. To do so I used a for loop to go through the elements in a and add them to result using the ss\_add() function from earlier. For b, I also used a for loop to go through all the elements in b, however before adding them to result, I used the ss\_contained() function to check whether any of the elements in b are already in result. If this is false, then the element is added. A boolean flag is used to indicate when the function is over, and if it were to fail false is returned, otherwise true is returned.

ii.) bool ss\_intersection

The use of this function is to see which elements are common in both a and b, and add them to result. I used a nested for loop to go through the elements in a, and compare each element to the elements in b. This is done by comparing the first element in a, with each element in b, and so forth. Once an element is found to be the same, it is added to result via the ss\_add() function. A boolean flag is used again to indicate when the function is over, and if it were to fail, false is returned, otherwise true is returned.

iii.) bool ss\_difference

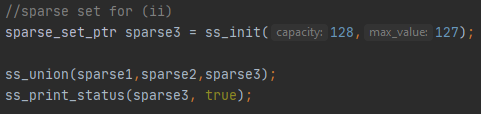
The use of this function is to add the elements which are only found in a, not in b. These elements are to be then added to result. I used a for loop to go through the elements in a, and then used the ss\_contained() function to check whether the element is in b. If not, then the ss\_add() function is used to add it to result. A boolean flag is used to indicate when the function is over. If it were to fail, false is returned, otherwise true is returned.

d.)

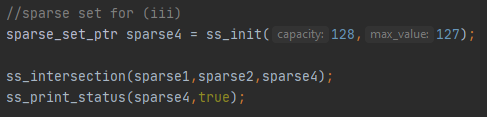
i.) We were required to create 2 sparse sets which I named “sparse1” and “sparse2”, and we had to set their max\_value as 127. I did so by declaring a buffer for each, and using the ss\_init() function. I then used for loops to add the elements into the sparse sets. We had to use the ss\_print\_status() function after each sub-task:



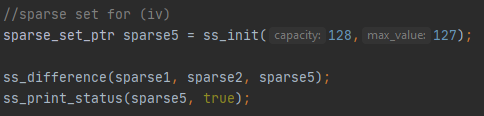
ii.) I created a new sparse set which I named “sparse3”, and used the ss\_union() function from before to get all the characters from both sets. I then used the ss\_print\_status() as required.



iii.) I created a new sparse set which I named “sparse4”, and used the ss\_intersection() function from before to obtain all the common characters used in both sets. I then used the ss\_print\_status() as required.



iv.) I created a new sparse set which I named “sparse5”, and used the ss\_difference() function from before to obtain all the characters that are in the first set but not in the second set. I then used the ss\_print\_status() as required.



v.) bool ss\_populate\_from\_file( const char \*filename, sparse\_set\_ptr result)

The use of this function is to populate the sparse set “result” from a file. To do so I used the fopen() function to obtain a file, and used “r” to indicate that the file was to be obtained to simply read the contents. I used an if statement for when the file is NULL which returns false, otherwise the contents of the file are looked through and implemented into the sparse set “result” and true is returned.