**FINALGROUP ASSIGNMENT**

**V1**

**CRP101NKK**

**GROUP – V**

**MEMBERS**

Farzad Sultani (LEADER) - fsultani4@myseneca.ca

LESVINE JOSEPH POOVAKULATH - [llesvine-joseph-poov@myseneca.ca](mailto:llesvine-joseph-poov@myseneca.ca)

## **Introduction**

We have worked on the final project by completing testing and coding on the given code which includes important functions

* Fundamentals
* Manipulations
* Tokenizing
* Conversions

FARZAD SULTANI – COMENTING

LESVINE JOSEPH POOVAKUTH - TESTING

## **Project Working**

The main function simply calls the function which does certain string activities

* Fundamentals – This function is used to find the character in the user inputted index.
* Manipulations – This function simply connects two user inputted strings to one string
* Tokenizing – This function separates user inputted space delaminated strings to individual separate characters
* Conversions – This function simply converts numeric characters to integers

**GITHUB LINK FOR COMMENTING**

[**https://github.com/Farzad911/supreme-engine/search?l=c**](https://github.com/Farzad911/supreme-engine/search?l=c)

**main.c**

#include "fundamentals.h"

#include "manipulating.h"

#include "converting.h"

#include "tokenizing.h"

int main() {

fundamentals();

manipulating();

converting();

tokenizing();

return 0;

}

## **tokenizing.h**

// that is the converting module header

#ifndef \_TOKENIZING\_H\_

#define \_TOKENIZING\_H\_

// this is the libraries

#include <stdio.h>

#include <string.h>

// Prototype Function

// This is the function that we use when we aply a convertion

void tokenizing(void); // Purpose: This function takes a user inputted string and tokenizes it

#endif

## **tokenizing.c**

//TOKENIZING MODULE SOURCE

#define \_CRT\_SECURE\_NO\_WARNINGS

#define BUFFER\_SIZE 300

// Includes all the necessary functions from stdio.h, string.h, and stdlib.h needed to manipulate

// strings and convert a string to an integer, a double, and a long value.

#include "tokenizing.h" //including header file

void tokenizing(void) {

// V1

printf("\*\*\* Start of Tokenizing Words Demo \*\*\*\n"); //printing statement for beginning the module

char words[BUFFER\_SIZE];// declaring array of char type

char\* nextWord = NULL;// declaring array of char type

int wordsCounter;//declaring variable of int type

// It start a do-while loop

do {

printf("Type a few words separated by space (q - to quit):\n"); // printing instructionstatement

fgets(words, BUFFER\_SIZE, stdin);// Prompt user for a integer numeric string

words[strlen(words) - 1] = '\0'; // replaces the new line at the end of the string with null to indicate when the string ends

if (strcmp(words, "q") != 0) { // only executes if user did not input 'q'

nextWord = strtok(words, " "); // use an empty space as the delimiter to know when to split the string into tokens

wordsCounter = 1; // intialize wordsCounter to 1, and start counting from 1 to keep track of how many words are being tokenized

// use a loop to print out the string, while incrementing wordsCounter to keep track of the number of words being tokenized

while (nextWord) {

printf("Word #%d is \'%s\'\n", wordsCounter++, nextWord);

nextWord = strtok(NULL, " "); // when the next value of the string reads as null, we break out of the loop\* /

}

}

} while (strcmp(words, "q") != 0); // breaks the loop if user inputs 'q' as the only character

// The do-while loop end

printf("\*\*\* End of Tokenizing Words Demo \*\*\*\n\n");// Output an ending statement to mark end of the module

}

/\* Version 2 \*/

printf("\*\*\* start of Tokenizing Phrases Demo \*\*\*\n");

char phrases [BUFFER\_SIZE];

char \* nextPhrase = NULL;

int phrasesCounter;

do {

printf("Type a few phrases separated by comma(q – to quit):\n”);

fgets (phrases, BUFFER\_SIZE, stdin);

phrases [strlen (phrases) -1]='\0';

if ((strcmp(phrases, "q") != 0)) {

nextPhrase = strtok (phrases, ",");

phrasesCounter = 1;

while (next Phrase) {

printf("Phrased #%d is \’s\’n", phrasesCounter++, nextPhrase);

nextPhrase = strtok (NULL, ",");

}

}

} while (strcmp(phrases, "q") != 0);

printf("\*\*\* End of Tokenizing Phrases Demo \*\*\*\n\n");

/\* Version 3 \*/

/\* Delimiter is now a single dot (".") \*/

printf("\*\*\* Start of Tokenizing Sentences Demo \*\*\*\n"); //printing statement for beginning the module

char sentences[BUFFER\_SIZE]; // declaring array of char type

char\* nextSentence = NULL;// declaring array of char type

int sentencesCounter;// declaring array of int type

// start a do-while loop

do {

printf("Type a few sentences separated by dot (q - to quit):\n"); // printing instruction statement

fgets(sentences, BUFFER\_SIZE, stdin);

sentences[strlen(sentences) - 1] = '\0'; // replaces the new line at the end of the string with null to indicate when the string ends

if ((strcmp(sentences, "q") != 0)) { // only executes if user did not input 'q'

nextSentence = strtok(sentences, "."); // use a dot as the delimiter to know when to split the string into tokens

sentencesCounter = 1; // intialize sentencesCounter to 1, and start counting from 1 to keep track of how many sentences are being tokenized

// use a loop to print out the string, while incrementing phrasesCounter to keep track of the number of sentences being tokenized

while (nextSentence) {

printf("Sentence #%d is \'%s\'\n", sentencesCounter++, nextSentence);

nextSentence = strtok(NULL, "."); // when the next value of the string reads as null, we break out of the loop

}

}

} while (strcmp(sentences, "q") != 0); // breaks the loop if user inputs 'q' as the only character

// The do-while loop end

printf("\*\*\* End of Tokenizing Sentences Demo \*\*\*\n\n"); // Output an ending statement to mark end of the module

}

## **manipulating.h**

// that is the converting module header

#ifndef \_MANIPULATING\_H\_

#define \_MANIPULATING\_H\_

// this is the libraries

#include <stdio.h> // For input/output and printing

#include <string.h> // For string manipulation functions

// This is the function that we use when we aply a convertion

void manipulating(void);

/\* Purpose: This function contains 2 input strings \*/

#endif

## **manipulating.c**

// CONVERTING MODULE SOURCE

#define \_CRT\_SECURE\_NO\_WARNINGS

#define BUFFER\_SIZE 80

#include "manipulating.h" //including header file

/\* Contains function prototypes and library declarations \*/

// V1

void manipulating(void) {

/\* Purpose: This function concatenates, compares and checks occurence bet. 2 input strings \*/

printf("\*\*\* Start of Concatenating Strings Demo \*\*\*\n");//printing statement for beginning the module

char string1[BUFFER\_SIZE];// declaring array of char type

char string2[BUFFER\_SIZE];// declaring array of char type

//Start a do-while loop

do {

printf("Type the 1st string (q - to quit):\n"); // printing instructionstatement

fgets(string1, BUFFER\_SIZE, stdin); // Get the 1st string input

string1[strlen(string1) - 1] = '\0';// Add the null terminator in the end of the string

if ((strcmp(string1, "q") != 0)) { // If the input is not equal to "quit"

printf("Type the 2nd string:\n");

fgets(string2, BUFFER\_SIZE, stdin);// Get the 2nd string input

strcat(string1, string2);

printf("Concatenated string is \'%s\'\n", string1);

}

} while (strcmp(string1, "q") != 0);// Repeat input as long as input is not equal to "quit"

// The do-while loop end

printf("\*\*\* End of Concatenating Strings Demo \*\*\*\n\n"); // Output an ending statement to mark end of the module

}

// V 2

printf("\*\*\* start of Comparing Strings Demo \*\*\*\n");

char compare1 [BUFFER\_SIZE];

char compare2 [BUFFER\_SIZE];

int result;

do {

printf("Type the 1st string to compare (q to quit):\n");

fgets (compare1, BUFFER\_SIZE, stdin);

compare1[strlen (comparel) – 1] = '\0';

if (strcmp(compare1, "q") != 0)

{

printf("Type the 2nd string to compare:\n");

fgets (compare2, BUFFER\_SIZE, stdin);

compare2[strlen (compare2) – 1] = "\0";

result = strcmp (comparel, compare2);

if (result < 0)

printf("\’%s\’ string is less than \'%s\'\n", compare1, compare2);

else if (result == 0)

printf("\’%s\’ string is equal to \'%s\'\n", compare1, compare2);

else

printf("\’%s\’ string is greater than \’%s\’\n", compare1, compare2);

}

} while (strcmp(compare1, "q") != 0);

printf("\*\*\* End of Comparing Strings Demo \*\*\*\n\n");

/\* Version 3 \*/

printf("\*\*\* Start of Searching Strings Demo \*\*\*\n");//printing statement for beginning the module

char haystack[BUFFER\_SIZE];//declaring array of char type

char needle[BUFFER\_SIZE];//declaring array of char type

char\* occurence = NULL;//declaring array of char type

//Start a do-while loop

do {

printf("Type the string (q - to quit):\n"); // printing instructionstatement

fgets(haystack, BUFFER\_SIZE, stdin); // Get the 1st string input

haystack[strlen(haystack) - 1] = '\0'; // Add the null terminator in the end of the string

if (strcmp(haystack, "q") != 0) { // Check if the user wants to exit the program

printf("Type the substring:\n");

fgets(needle, BUFFER\_SIZE, stdin); // Get the 2nd string input

needle[strlen(needle) - 1] = '\0'; // Add the null terminator in the end of the string

occurence = strstr(haystack, needle); // Check if 2nd string occurs on 1st string

if (occurence) // If found, return the position where 2nd string occurs

printf("\'%s\' found at %d position\n", needle, (int) (occurence - haystack));

else

printf("Not found\n");

}

} while (strcmp(haystack, "q") != 0); // Continue while user doesn't want to quit

// The do-while loop end

printf("\*\*\* End of Searching Strings Demo \*\*\*\n\n"); // Output an ending statement to mark end of the module

}

**fundementals.h**

// that is the converting module header

#ifndef \_FUNDAMENTALS\_H\_

#define \_FUNDAMENTALS\_H\_

// this is the libraries

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//V1 Takes non-empty string and a number as input and gives the character at that number(index) as output.

// This is the function that we use when we aply a convertion

void fundamentals(void);

#endif

## **fundementals.c**

// Start macros and headers here

#define \_CRT\_SECURE\_NO\_WARNINGS

#define BUFFER\_SIZE 80

#define NUM\_INPUT\_SIZE 10

#include "fundamentals.h"//including header file

// End macros and headers

void fundamentals(void) {

// V1

printf("\*\*\* start of Indexing Strings Demo \*\*\*\n");//printing statement for beginning the module.

char buffer1[BUFFER\_SIZE];// declaring array of char type

char numInput[NUM\_INPUT\_SIZE]; // declaring array of char type

size\_t position;

// It start a do-while loop

do {

printf("Type not empty string (q - to quit): \n"); // printing instructionstatement

fgets (buffer1, BUFFER\_SIZE, stdin); // Prompt user for a integer numeric string

buffer1[strlen (buffer1) - 1] = '\0'; // Replaces user entered newline with null terminator

// checks if entered string is not equals to 'q'.

if (strcmp (buffer1, "q") != 0) {

printf("Type the character position within the string: \n");

fgets (numInput, NUM\_INPUT\_SIZE, stdin);// Prompt user for a integer numeric string

numInput[strlen (numInput) - 1] = '\0'; // Replaces user entered newline with null terminator

position = atoi (numInput); // Converts entered string to integer.

// Checks if converted integer is larger then string length and assigns max position if true.

if (position >= strlen (buffer1)) {

position = strlen (buffer1) - 1;

printf("Too big... Position reduced to max. availbale\n");

}

printf("The character found at %d position is '%c\'\n",(int)position, buffer1[position]);

}

} while (strcmp (buffer1, "q") != 0); // breaks the loop if user inputs 'q' as the only character

// The do-while loop end

printf("\*\*\* End of Indexing Strings Demo \*\*\*\n\n"); // Output an ending statement to mark end of the module

// V 2

printf("\*\*\* Start of Measuring Strings Demo \*\*\*\n");

char buffer2 [BUFFER\_SIZE];

do {

printf("Type a string (q - to quit): \n");

fgets (buffer2, BUFFER SIZE, stdin);

buffer2 [strlen (buffer2) - 1] = '\0';

if (strcmp(buffer2, "q") != 0)

printf("The length of \'%s\' is %d characters\n",buffer2, (int) strlen (buffer2));

} while (strcmp (buffer2, "q") != 0);

Printf("\*\*\* End of Measuring Strings Demo \*\*\*\n\n");

// V3

printf("\*\*\* Start of Copying Strings Demo \*\*\*\n"); //printing statement for beginning the module.

char destination [BUFFER\_SIZE]; // declaring array of char type

char source[BUFFER\_SIZE]; // declaring array of char type

// Start a do-while loop

do {

//Replaces first element of destination with null terminator

destination [0] = '\0';

printf("Destination string is reset to empty\n"); // printing instruction statement

printf("Type the source string (q - to quit): \n"); // printing instruction statement

fgets (source, BUFFER\_SIZE, stdin); //Gets the user entered string

//Replaces user entered new line with null terminator

source[strlen (source) - 1] = '\0';

// checks if entered string is not equals to 'q'.

if (strcmp (source, "q") != 0) {

//copies string from source to destination

strcpy (destination, source);

printf("New destination string is \'%s\'\n", destination);

}

//loops till user enter q

} while (strcmp (source, "q") != 0); // breaks the loop if user inputs 'q' as the only character

// The do-while loop end

printf("\*\*\* End of Copying Strings Demo \*\*\*\n\n"); // Output an ending statement to mark end of the module

}

## **converting.c**

#include "converting.h"

void converting() {

/\* Version 1 \*/

// CONVERTING MODULE SOURCE

#define \_CRT\_SECURE\_NO\_WARNINGS

#define BUFFER\_SIZE 80

// Includes all the necessary functions from stdio.h, string.h, and stdlib.h needed to manipulate

// strings and convert a string to an integer, a double, and a long value.

#include "converting.h" //including header file

// V1

void converting(void) {

printf("\*\*\* Start of Converting Strings to int Demo \*\*\*\n"); //printing statement for beginning the module.

char intString[BUFFER\_SIZE];// declaring array of char type

int intNumber;//declaring variable of int type

// It start a do-while loop

do {

printf("Type an int numeric string (q - to quit) : \n"); // printing instructionstatement

fgets(intString, BUFFER\_SIZE, stdin);// Prompt user for a integer numeric string

intStrings[strlen(intString) - 1] = '\0';// Replace new line with null terminator

if (strcmp(intString, "q") != 0) { // Check user input, exit if 'q' was entered.

intNumber = atoi(intString);// Convert user entry from string to integer value.

printf("Converted number is %d\n", intNumber);//printing convertednumber

}

} while (strcmp(intString, "q") != 0); // breaks the loop if user inputs 'q' as the only character

// The do-while loop end

printf("\*\*\* End of Converting Strings to int Demo \*\*\*\n\n");// Output an ending statement to mark end of the module

}

/\* Version 2 \*/

printf("\*\*\* start of Converting Strings to double Demo \*\*\*\n");

char doublestring[BUFFER\_SIZE];

double doubleNumber;

do {

printf("Type the double numeric string (q to quit):\n");

fgets (doublestring, BUFFER\_SIZE, stdin);

doublestring [strlen (doubleString) -1] = '\0';

if ((strcmp (doublestring, "q") != 0)) {

doubleNumber = atof (doublestring);

printf("Converted number is %f\n", doubleNumber);

} while (strcmp (doubleString, "q") != 0);

printf("\*\*\*End of Converting Strings to double Demo \*\*\n\n");

/\* Version 3 \*/

printf("\*\*\* Start of Converting Strings to long Demo \*\*\*\n");//printing statement for beginning the module.

char longString[BUFFER\_SIZE];// declaring array of char type

long longNumber;//declaring variable of long type

// start of a do-while loop

do {

// Prompt user for a long numeric string, replacing the new line read by

// fgets with a null terminator.

printf("Type a long numeric string (q - to quit):\n");// printing instructionstatement

fgets(longString, BUFFER\_SIZE, stdin);// Prompt user for a integer numeric string

longString[strlen(longString) - 1] = '\0';

// Check user input, exit if 'q' was entered.

if ((strcmp(longString, "q") != 0)) {

// Convert user entry from numeric string to a long, then display it

longNumber = atol(longString);

printf("Converted number is %ld\n", longNumber);

}

} while (strcmp(longString, "q") != 0);// breaks the loop if user inputs 'q' as the only character

// The do-while loop end

printf("\*\*\* End of Converting Strings to long Demo \*\*\*\n\n");// Output an ending statement to mark end of the module

}

## **converting.h**

// that is the converting module header

#ifndef \_CONVERTING\_H\_

#define \_CONVERTING\_H\_

// this is the libraries

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Prototype Function

// This is the function that we use when we aply a convertion

void converting(void);

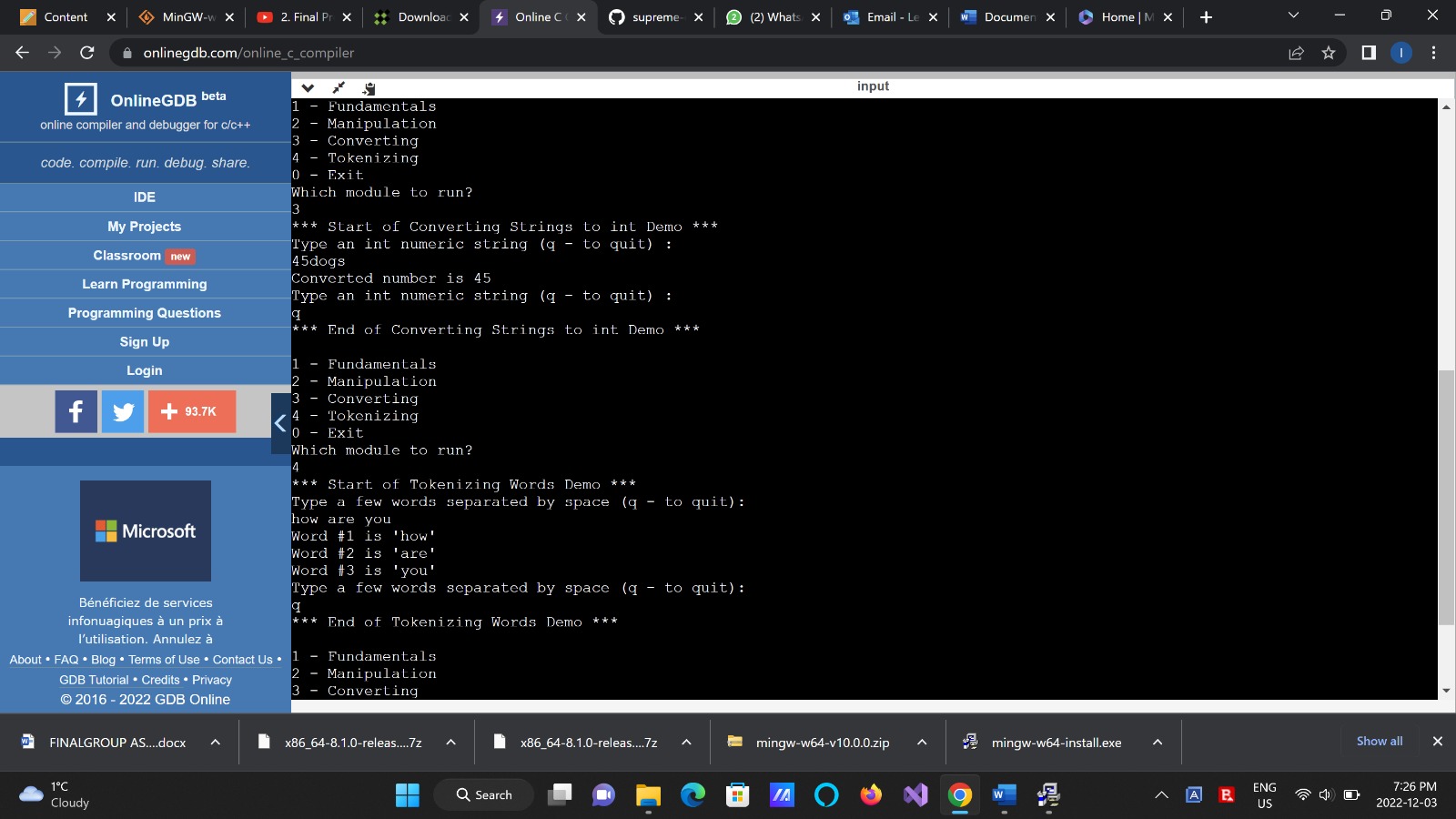
#endif

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program or module:** | **main** | **Save this worksheet in an Excel file named main\_test\_cases.xlsx** | | | | |
| **Version 1 Basic Test Cases to confirm module functionality when integrated into an application.** | | | | | **Run by: Date:** | *Tester Date* |
| **Description** | **+ / − Purpose** | **Data Input** | **Expected Output** | **Actual output if unexpected** | **Success?** | **Comments** |
| record name | to capture name of person running tests in main\_testing.txt | *your name* |  |  | n/a |  |
| Fundamentals | typical case | if tested withing the length of string | returns the characted in that position |  | yes | code works fine if checked within the lenth of string |
| Fundamentals | maximal edge case | if tested more than the length of string | error message that search index more than the length of string and a return of the last element of the string |  | yes |  |
| Fundamentals | minimal edge case | if tested less than the length of string | error message that search index more than the length of string and a return of the last element of the string | error messaage should be search index less than length of string | no | code if changed, to accept a negative index and promts seach index less than string length |
| Fundamentals | minimal edge case | can't input a characted "q" | quits out of the loop | even though the output is expected, it is an error | yes | code return's the expected o/p but code could me modified to accept "q" and an other method to exit out of the loop |
| Manipulations | minimal edge case | can't input a characted "q" | quits out of the loop | even though the output is expected, it is an error | yes | code return's the expected o/p but code could me modified to accept "q" and an other method to exit out of the loop |
| Manipulations | typical case | if 2 string inputed, | join's both the string return the concated string |  | yes |  |
| Manipulations | maximal edge case | if 2 string inputed, | join's both the string return the concated string along with a newline character |  | no | the new line character can't be modified |
| Tokenizing | minimal edge case | if started with a space | ignores the 1st space and continues to lokk for the next character | the space characted is ignores | no | code can be modified to accept a space in the beginning |
| Tokenizing | typical case | if strings are inputed by space delimination | displays each string seperately |  | yes |  |
| Tokenizing | maximal edge case | if the 1st characted is a "q", | code exits |  | yes | expected output but code can be modified to accept "q" |
| Tokenizing | maximal edge case | if the 1st characted is a "q", followed by a different character | code exits after printing out the output |  | yes |  |
| Conversions | minimal edge case | if a string with numeric character are present in the end of the inputed string | code displays 0 as the integer | code should read the numeric character no matter the position | no | code should be modified so that it can read numeric character in which ever position it is located |
| Conversions | typical case | if a string with numeric character are present in the beginning | code converts the numeric character to integer |  | yes |  |
| Conversions | maximal edge case | if the 1st characted is a "q", | code exits |  | yes | expected output but code can be modified to accept "q" |
| **Version 2 Basic Test Cases to confirm module functionality when integrated into an application.** | | | | | **Run by: Date:** |  |
| Fundamentals | minimal edge case |  |  |  |  |  |
| Fundamentals | typical case | if a string is inputed, | the length of string gets printted out |  | yes | the code works fine if inputted within the buffer length |
| Fundamentals | maximal edge case | doesn’t accept character q as it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |
| Manipulations | minimal edge case |  |  |  |  |  |
| Manipulations | typical case | if 2 strings are inputted, | it compares the ASCI values of characters and returns the comared result |  | yes | the code works fine if inputted within the buffer length |
| Manipulations | maximal edge case | doesn’t accept character q as it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |
| Tokenizing | minimal edge case |  |  |  |  |  |
| Tokenizing | typical case | if coma seperated values are entered, | it returns all the comma seperated values as separate strings |  | yes | the code works fine if inputted within the buffer length |
| Tokenizing | maximal edge case | doesn’t accept character q if it’s the 1st stringas it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |
| Conversions | minimal edge case |  |  |  |  |  |
| Conversions | typical case | if a double character is inputed | it returns the double value in the double data type |  | yes | the code works fine if inputted within the buffer length |
| Conversions | maximal edge case | if a double character followed by a alphabet is inputed, | it takes the double value and ignores the alphabet | it should have printed out an error message | no | the code should be modified for inputs like (45.k) instead of printing out error message, its displays 45.00000 |
| Fundamentals | minimal edge case |  |  |  |  |  |
| **Version 3 Basic Test Cases to confirm module functionality when integrated into an application.** | | | | | **Run by: Date:** |  |
| **Description** | **+ / − Purpose** | **Data Input** | **Expected Output** | **Actual output if unexpected** | **Success?** |  |
| Fundamentals | minimal edge case |  |  |  |  |  |
| Fundamentals | typical case | if a string under buffer\_size | copy's the inputed string to an other string and prints out the new string |  | yes | the code works as expected |
| Fundamentals | maximal edge case | doesn’t accept character q as it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |
| Manipulations | minimal edge case |  |  |  |  |  |
| Manipulations | typical case | a string is inputted and a substring is inputed | if the substring is not in the string, it outputs 'not found' otherwise prints out the index of substring |  | yes | the code works as expected |
| Manipulations | maximal edge case | doesn’t accept character q as it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |
| Tokenizing | minimal edge case |  |  |  |  |  |
| Tokenizing | typical case | if string with dot is typed in, | the string is broken to substring based on the position of dot |  | yes | the code works as expected |
| Tokenizing | maximal edge case | doesn’t accept character q as it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |
| Conversions | minimal edge case | if a character string is inputted 1st | the code prints out 0 | it should iterate throughout the string and return the numeric value | no | the code could be modified so that it accepts numeric character no matter the position of the numeric string |
| Conversions | typical case | if a numeric string is inputted | the string is converted to a integer |  | yes | the code works as expected |
| Conversions | maximal edge case | doesn’t accept character q as it is set to be the exiting character | the code exits |  | no | the code can be modified to accept q as a chaarcter |

**VERSION – 1**

A screenshot of a computer

Description automatically generated



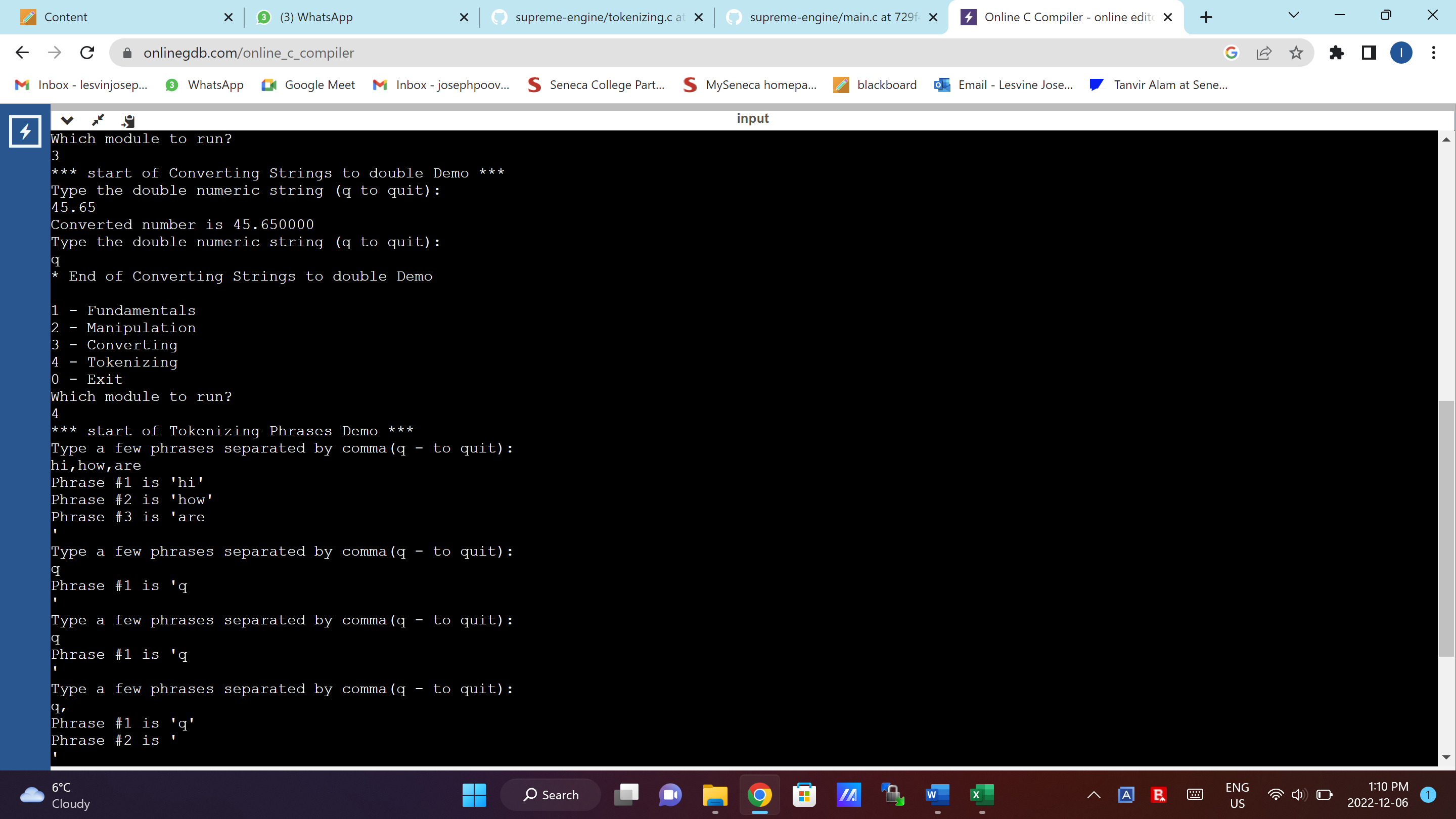
A screenshot of a computer

Description automatically generated

**VERSION – 2**

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

**VERSION – 3**

A screenshot of a computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**CONCLUSION**

The complete working of code can be easily understood from the commenting and almost all possible test cases were considered and outputs were recorded.