CSE240 - C/C++ Assignment #1

50 points

Topics:

- Created and used variables
- Used printf and scanf to work with input and output
- Use simple File I/O
- Used control structures to control the flow of logic
- Work with math and random numbers
- Created these things from scratch
- Create a some basic void and value returning functions

Description

This assignment is going to get you used to working with the Unix/Linux compiler through some exercises and translate Morse Code through file I/O.

Use the following Guidelines:

- Give identifiers semantic meaning and make them easy to read (examples numStudents, grossPay, etc).
- Keep identifiers to a reasonably short length.
- User upper case for constants. Use title case (first letter is upper case) for classes. Use lower case with uppercase word separators for all other identifiers (variables, methods, objects).
- Use tabs or spaces to indent code within blocks (code surrounded by braces). This includes classes, methods, and code associated with ifs, switches and loops. Be consistent with the number of spaces or tabs that you use to indent.
- Use white space to make your program more readable.
- **DO NOT USE** #include <string> for this assignment we are working in C-Style string. You **may not** use the string data type in C++.
    - #include <string.h> (the C-style string library) is permitted though

Important Note:

All submitted assignments must begin with the descriptive comment block. To avoid losing trivial points, make sure this comment header is included in every assignment you submit, and that it is updated accordingly from assignment to assignment.

```
/*
Author: <your name here>
Date: <date>
Description: <short description of code file>
Usage: <usage syntax>
*/
```

Programming Assignment:

Instructions:

This assignment has three parts:

1. Ungraded exercises you should do to become familiar with Unix/Linux
2. Ungraded exercises to familiarize yourself with C/C++, GDB and strtok()
3. Programming assignment

The first two parts are there to help you through understanding how to connect with and use Unix/Linux on general.asu.edu.

For these exercises you should use and SSH client to connect to general.asu.edu and text editor on general.asu.edu.

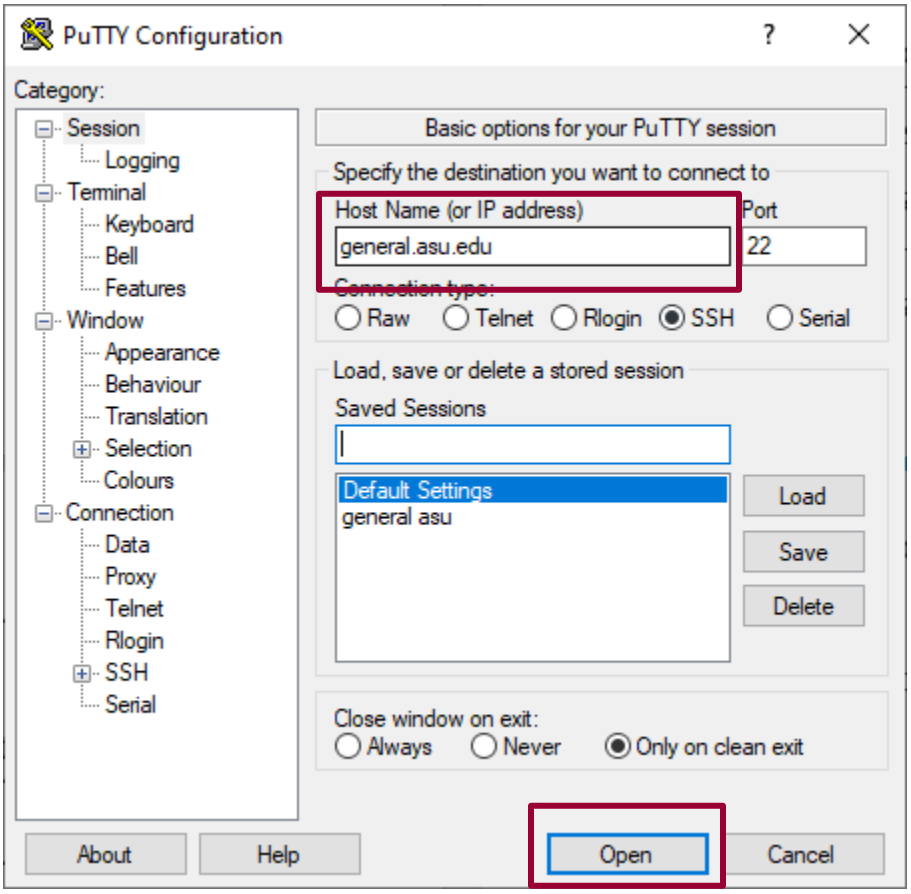
Specifications:

Set-Up

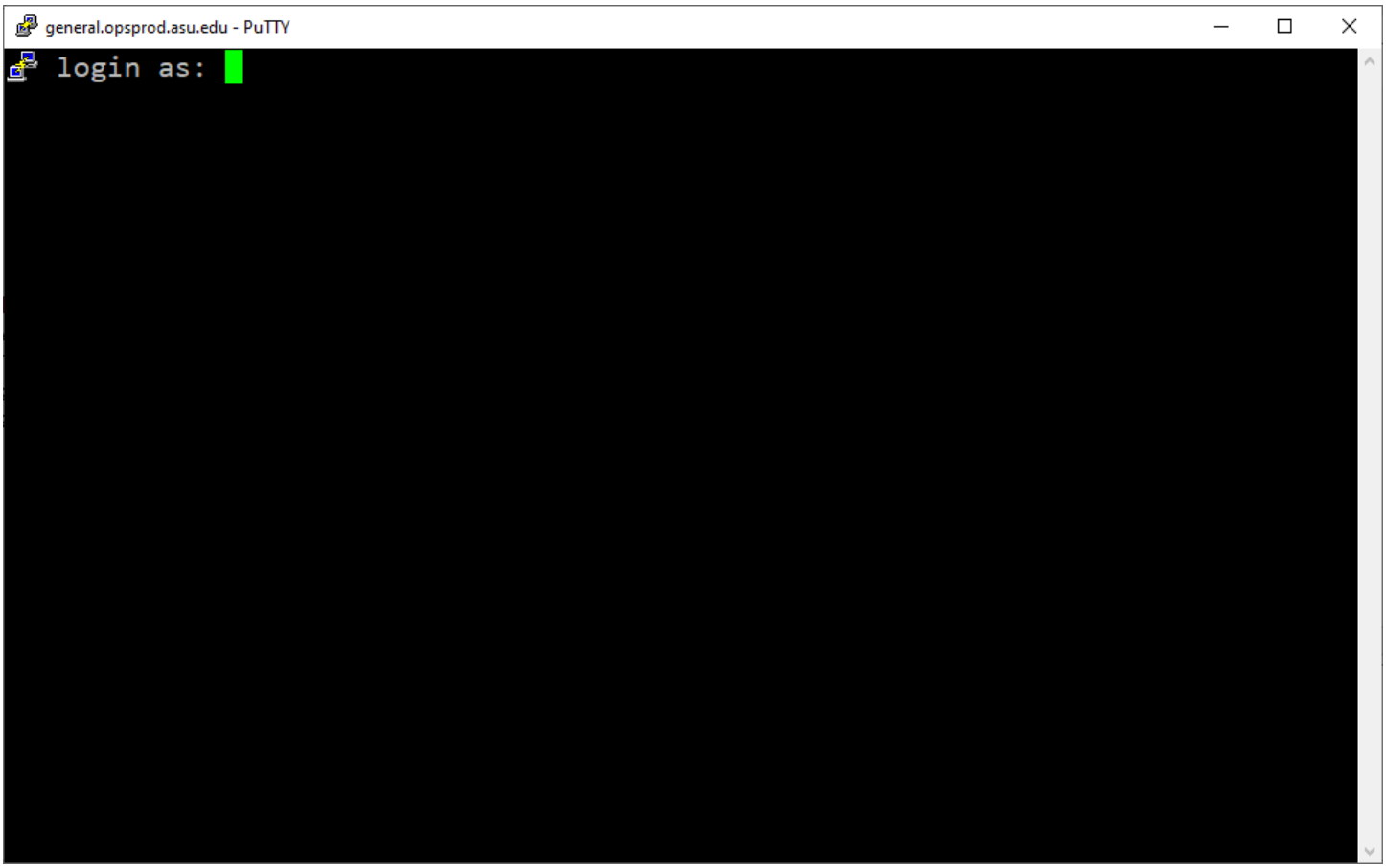
If you are on Windows:

Download an SSH client of your choosing. I recommend Putty (www.putty.org).

Using putty, connect to general.asu.edu



Once connected enter your username (ASURiteID) and password. NOTE your password will be invisible.



If you are on a Mac:

You can connect to general.asu.edu through the built in SSH client.

Open your terminal and enter the command:

ssh <your ASURiteID>@general.asu.edu

Once connected, you will be prompted for your password.

Sorry for the lack of screenshots here, I don't have a Mac

Part 1 – Getting used to Unix/Linux, C/C++ compiler (NON-GRADED – do it anyway!)

Math Loop:

On general.asu.edu, use vi, emacs or nano to enter the following code into a file named: <your last name here>_mathloop.c (example: selgrad_mathloop.c)

```
#include <stdio.h>
void main()
{
	char ch;
	ch = '+';
	int f, a = 10, b = 20;
	printf("ch = %c\n", ch);
	switch (ch) {
		case '+': f = a + b; printf("f = %d\n", f);
		case '-': f = a - b; printf("f = %d\n", f);
		case '*': f = a * b; printf("f = %d\n", f);
		case '/': f = a / b; printf("f = %d\n", f);
		default: printf("invalid operator\n");
	}
	ch = '-';
	a = 10, b = 20;
	printf("ch = %c\n", ch);
	switch (ch) {
		case '+': f = a + b; printf("f = %d\n", f);
		case '-': f = a - b; printf("f = %d\n", f);
		case '*': f = a * b; printf("f = %d\n", f);
		case '/': f = a / b; printf("f = %d\n", f);
		default: printf("invalid operator\n");
	}
	ch = '*';
	a = 10, b = 20;
	printf("ch = %c\n", ch);
	switch (ch) {
		case '+': f = a + b; printf("f = %d\n", f);
		case '-': f = a - b; printf("f = %d\n", f);
		case '*': f = a * b; printf("f = %d\n", f);
		case '/': f = a / b; printf("f = %d\n", f);
		default: printf("invalid operator\n");
	}
	ch = '/';
	a = 10, b = 20;
	printf("ch = %c\n", ch);
	switch (ch) {
		case '+': f = a + b; printf("f = %d\n", f);
		case '-': f = a - b; printf("f = %d\n", f);
		case '*': f = a * b; printf("f = %d\n", f);
		case '/': f = a / b; printf("f = %d\n", f);
		default: printf("invalid operator\n");
	}
	ch = '%';
	a = 10, b = 20;
	printf("ch = %c\n", ch);
	switch (ch) {
		case '+': f = a + b; printf("f = %d\n", f);
		case '-': f = a - b; printf("f = %d\n", f);
		case '*': f = a * b; printf("f = %d\n", f);
		case '/': f = a / b; printf("f = %d\n", f);
		default: printf("invalid operator\n");
	}
}
```

Compile and run this code and observe its behaviors.

```
gcc –g –Wall <lastname>_mathloop.c –o mathloop
```

REMEMBER you need to use ./ to execute your code. For example if I compile hello.c into the executable hello (gcc -g -Wall hello.c -o hello) … I run hello via the command ./hello

The code provided is clearly incorrect. It has Semantic errors – such as the missing break statements in the switch. Repair the semantic errors, compile and run the code again.

Now let's improve this code.

Improving the Algorithm Logic:

Instead of repeating the same code five times – **modify** the code to use a loop to ask for 5 user inputs.

While loops in C are the same as they are in java:

```
while(<conditional>)
{
//do stuff
}
```

For-loops in C don't allow for creation of a variable in the initialization
You can't do: for(int c = 0; …) in C you have to declare your variable elsewhere:

```
int c;

for(c = 0; …)
```

Use an input statement such as `scanf("%c", &ch);` to replace the assignment statements such as ch = '+';

Warning – remember you can leave a \n character in the input buffer. You might need to sanitize your inputs. The scanf command as given will read only one character.

The easiest way to do this is with scanf("% c", &ch); but that's not always reliable as not every compiler seems to accept it. It should be fine though.

Transfer the edited .c file to your computer via an FTP client, WinSCP or https://webapp4.asu.edu/myfiles/app

Using command line debugger ***(important to learn)***

You are given a file “gdbChallenge.c”

Rename this file: <lastname>_gdb.c

Compile the code: gcc -g -Wall <lastname>_gdb.c -o challenge

Run the executable: ./challenge

You will get nothing but 0’s in your output. Clearly this code has problems! Time to find the error and fix it.

Execute the command: gdb ./challenge

This will load the executable into Gnu Debugger (gdb). Gdb is a command line debugger, but it’s still quite powerful.

You can look at the code for the executable by entering the command “list”. List will show you code listings sequentially, to start over from the top or a line number enter the command: list <line number>

Or to start from the top: list 0

Set a break point by entering the command: b 10

This will pause execution at that line.

Now run the code with the command: run

The code will stop at line 10 (the beginning of the for-loop)

```
Breakpoint 1, main (argc=1, argv=0x7fffffffe2d8) at gdbChallenge.c:10
10          for(i = 0; i < 10000; i++)
```

To proceed to the next line simply enter the command: next

You should now see:

```
(gdb) next
12              int value = (int)frand() * 10000;
```

Let's get some information about what's going on… value and i are both meaningful to us. So enter the commands:

display i

display value

Next time you type "next" you should see:

```
13              printf("%d\n",value);
2: value = 0
1: i = 0
```

Keep hitting enter for a while – this will repeat your "next" command automatically

```
(gdb)
0
10          for(i = 0; i < 10000; i++)
1: i = 0
(gdb)
12              int value = (int)frand() * 10000;
2: value = 0
1: i = 1
(gdb)
13              printf("%d\n",value);
2: value = 0
1: i = 1
(gdb)
0
10          for(i = 0; i < 10000; i++)
1: i = 1
(gdb)
12              int value = (int)frand() * 10000;
2: value = 0
1: i = 2
(gdb)
13              printf("%d\n",value);
2: value = 0
1: i = 2
(gdb)
0
10          for(i = 0; i < 10000; i++)
1: i = 2
(gdb)
12              int value = (int)frand() * 10000;
2: value = 0
1: i = 3
(gdb)
13              printf("%d\n",value);
2: value = 0
1: i = 3
(gdb)
```

```
0
10          for(i = 0; i < 10000; i++)
1: i = 3
(gdb)
12              int value = (int)frand() * 10000;
2: value = 0
1: i = 4
(gdb)
13              printf("%d\n",value);
2: value = 0
1: i = 4
(gdb)
0
10          for(i = 0; i < 10000; i++)
1: i = 4
(gdb)
12              int value = (int)frand() * 10000;
2: value = 0
1: i = 5
```

What can we notice? Value is remaining as 0 while i is incrementing properly through the for-loop.

We've identified our error! Value isn't being set properly.

Enter the command: kill

Respond with 'y' to the prompt.

Enter the command: quit

I've coded frand() to function like Math.rand() in Java. It will return a floating point value between 0 and 1. I want to use it to generate random integers between 0 and 10000 …

Edit the code to repair this error. Hint: I'm missing some parentheses.

Recompile the code.

Redo the gdb process to demonstrate to yourself that the code is working properly now.

***Yes**, I realize you can fix this code through visual debugging … that's not the point, the point is to familiarize you with GDB – don't cheat yourself out of the process*

Part 2 – Getting used to strtok()

The function strtok() comes from the library <string.h> which is a C-String library. It tokenizes a string into substrings based on a delimeter.

Function signature:

```
char *strtok(char *str, const char *delim)
```

The function takes two parameters:

1. A source string
    a. Since this is a character pointer, any character array can be passed here.
    b. Remember to be a C-String though it needs a '\0' at the end
2. A string literal that describes the delimiters
    a. There is no special syntax here, just put them right next to each other with no white-space (unless you want space to be a delimiter)

The function returns a character pointer (i.e. a string). The function should be called repeatedly until it returns NULL. Each time it is called it returns the next token.

When tokenizing a single string – the first time strtok() is called, you provide the source string. To keep tokenizing that string you pass NULL as the first parameter.

Let's parse some Lorem Ipsum:

```
Lorem ipsum dolor sit amet consectetur adipiscing elit Integer tempor posuere nibh quis tempor Sed
      eu turpis nulla Suspendisse vestibulum blandit enim vitae feugiat Nulla malesuada orci ut
      nisi blandit congue Maecenas vulputate magna in tellus eros
```

Sample Code:

Type in this code (don't just copy paste, MS Word might add non-code characters), you should be able to copy/paste the Lorem text from the previous page though.

```
#include <string.h>
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
	char sourceString[] = "Lorem ipsum dolor sit amet consectetur adipiscing elit Integer tempor
		posuere nibh quis tempor Sed eu turpis nulla Suspendisse vestibulum blandit enim vitae
		feugiat Nulla malesuada orci ut nisi blandit congue Maecenas vulputate magna in tellus
		eros";
	char* token;

	printf("Splitting string...");

	token = strtok(sourceString, " "); //first read

	while(token != NULL)
	{
		printf("%s\n", token);
		token = strtok(NULL, " "); //keep tokenizing notice NULL
	}
	return 0;
}
```

Output:

```
Splitting string...Lorem
ipsum
dolor
sit
amet
consectetur
adipiscing
elit
Integer
tempor
posuere
nibh
quis
tempor
Sed
eu
turpis
nulla
Suspendisse
vestibulum
blandit
enim
vitae
feugiat
Nulla
malesuada
orci
ut
nisi
blandit
congue
Maecenas
vulputate
magna
in
tellus
eros
```

Here's a modification of that code that processes a character at a time and uses a function, once again, don't just copy/paste:

```
#include <string.h>
#include <iostream>
using namespace std;

int charToInt(char);


int main(int argc, char** argv)
{
	char sourceString[] = "Lorem ipsum dolor sit amet consectetur adipiscing elit Integer tempor
		posuere nibh quis tempor Sed eu turpis nulla Suspendisse vestibulum blandit enim vitae
		feugiat Nulla malesuada orci ut nisi blandit congue Maecenas vulputate magna in tellus
		eros";
	char* token;

	printf("Splitting string...");

	token = strtok(sourceString, " ");

	while(token != NULL)
	{
		printf("%s\n", token);
		for(int i = 0; i < strlen(token); i++)
		{
			printf("Character value of %c = %d\n", token[i], charToInt(token[i]));
		}
		token = strtok(NULL, " ");

	}
	return 0;
}

int charToInt(char item)
{
	if((item >= 'a') && (item <= 'z'))
	{
		return (int)item;
	}
	else if ((item >= 'A') && (item <= 'Z'))
	{
		return (int)item;
	}
	else
		return -1;
}
```

Partial output:

```
Splitting string...Lorem
Character value of L = 76
Character value of o = 111
Character value of r = 114
Character value of e = 101
Character value of m = 109
ipsum
Character value of i = 105
Character value of p = 112
Character value of s = 115
Character value of u = 117
Character value of m = 109
dolor
Character value of d = 100
Character value of o = 111
Character value of l = 108
Character value of o = 111
Character value of r = 114
```

Part 3 – Programming Assignment [graded portion]

Part A – Characters to Morse Code Format

You are going to create a Morse Code translator in C/C++. You may use either C or C++ for this assignment.

This program will read an input file which will have a series of words using the English Alphabet characters a-z, A-Z. These words will be space delimited with no punctuation. Sentences will terminate with a newline.

A file might look something like this:

```
this is a test
this too is a test
sos
```

We will be using the International Standard of Morse Code:

```
A    .-
B    -...
C    -.-.
D    -..
E    .
F    ..-.
G    --.
H    ....
I    ..
J    .---
K    -.-
L    .-..
M    --
N    -.
O    ---
P    .--.
Q    --.-
R    .-.
S    ...
T    -
U    ..-
V    ...-
W    .--
X    -..-
Y    -.--
Z    --..
```

Remember, you should be handling both lowercase and uppercase letters the same. Characters ‘a’ and ‘A’ should translate to the same pattern “.-“.

The input file will consist of several lines of words with the maximum length of a line being 255 characters (including spaces). There can be any number of lines in a file.

Read the file a line at a time as a C-Style String (character array that ends in \0). Process through each character in the line to translate that character into the appropriate Morse Code pattern. Output that pattern to the output file.

Each character should be separated by a | and an actual space should be outputted as a /.

If you read in the line "the cat" the output would be:

"the cat" ➔ -|....|.|/|-.-.|.-|-

This is a Row-by-row, character-by-character algorithm. DO NOT try to be clever here. DO NOT just read a single character at a time from the file.

Part A Notes:

- Remember you are outputting the Morse Code to an output file!
  - You may also output it to stdout as well if you choose
- use some form of getline() to read an entire line from the file
  - C's getline() can take a file pointer as the source parameter
  - C++'s cin.getline can be used with your ifstream, if your ifstream is named infile you can do infile.getline()
- fscanf("%s") won't be helpful here because scanf delimits on spaces and newlines
- after reading a whole line from the file, you can use strtok() with space as your delimeter to split up the line into words if you want
  - process each word a character at a time
  - You can also just process the whole line a character at a time without tokenizing.
- The maximum line length will be 255 characters.

Part B – Translate Morse Code back into English Alphabet

In Part B, you're going to take a Morse Code file and translate it back into the English Alphabet.

This is "easy" if you use strtok() properly and strcmp() properly.

Reach the Morse Code file a line at a time. The line will be a maximum of ~1785 characters. *I think the worse the line can be is 255 characters that translate to a 4 pattern in morse code with a space between each character* ie: ----|/|

Output the English Character to the output file.

Part B Notes:

- Remember you're outputting to an output file
  - You may also output to stdout if you choose
- You really only have one delimeter to worry about here |
  - / represents a space, so handle it accordingly
- Look up strcmp in #include <string.h> trust me

Requirements:

Command line interface:

This program will not have any menu interface. It will work entirely as a command line utility.

Syntax: <exe> <input file> <-mc|-e> <output file>

Sample translate to morse code: ./morse message_orig.txt -mc message.mc

Sample translate to English: ./morse message.mc -e message.txt

If the user does not provide the proper command line arguments you should output an error message proving proper usage information!

Required Functions:

englishToMorse()

You should create a function called englishToMorse that takes a character as a parameter and returns a string literal representation as shown above.

If a character is passed in other than a-f, A-F or space; return “eeee”

Remember a space pass in should return “/”

Prototype: const char* englishToMorse(char);

Very simple input -> output function that will give you a short string literal that you can write to your output file. DO NOT pass the entire line to this function.

morseToEnglish()

You should create a function called morseToEnglish that takes const char* as a parameter and returns and English character as shown above.

If the pattern “eeee” is passed in the function should return a # symbol.

Remember a “/” passed in should return a space ‘ ‘.

Prototype: char morseToEnglish (const char*);

Very simple input -> output function that will give you a character that you can write to your output file. DO NOT pass the entire line to this function.

General Notes:

- Your program should be able to handle upper-case or lower-case letters
  - A-F, a-f
  - There is a simple trick you can do with a *switch* statement to make this easy
- When translating Morse Code back to English you can choose either upper- or lower-case characters for your output.
- You should not handle this by relying on a library call to force upper-case or lower-case
  - You may write YOUR OWN function to do this if you like though
- If the user doesn't run the program with proper command-line syntax, you should report an error and give them proper usage
- You should **not** be doing the file I/O one character at a time
  - For English->Morse:
    - read a whole line, tokenize, then process the tokens one character at a time
  - Morse->English
    - read a whole line, tokenize, then process the tokens
- Your output should have the same line structure as the input
  - i.e. newlines should be in the right place

Sample Output:

Example Input File #1:

```
abc def ghi
abc def ghi
abc def ghi
abc def ghi
abc def ghi
```

To Morse Command:

```
morse test.txt -mc testMC.mc
```

Output File:

```
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
```

To English Command:

```
morse testMC.mc -e testENG.txt
```

Output File:

```
ABC DEF GHI
ABC DEF GHI
ABC DEF GHI
ABC DEF GHI
ABC DEF GHI
```

Example Input File #2:

```
abc def 123
abc def ghi
abc def ghi
```

To Morse Command:

```
morse bad.txt -mc badMC.mc
```

Output File:

```
.-|-...|-.-.|/|-..|.|..-.|/|eeee|eeee|eeee
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
.-|-...|-.-.|/|-..|.|..-.|/|--.|....|..
```

To English Command:

```
morse badMC.mc -e badENG.txt
```

Output File:

```
ABC DEF ###
ABC DEF GHI
ABC DEF GHI
```

Code Hints:

- switch does exist in C & C++
- const char* is the data type for a string literal
- Even though there is only one required function, consider writing other functions that might help you out
- #include <string.h> is OK to use for functions such as strlen()
- Treat all the data as character – don't try to convert to numbers
- This is a very straight forward assignment involving if-statements, loops and file I/O …
    - If you're getting all messed up, ask yourself "Am I over-complicating this?" chances are you are
- Write your routines without command-line arguments stuff first!!
    - Get your core algorithms working with hard-coded file names
    - Once you know that your algorithms are working, deal with the command line arguments portion
- Don't forget to torture your code a little. Make sure it works with upper-case and lower-case, make sure it reacts properly to characters other than a-f, A-F, and space

Grading of Programming Assignment

The TA will grade your program following these steps:

(1) Compile the code. If it does not compile a U or F will be given in the Specifications section. This will probably also affect the Efficiency/Stability section.

(2) The TA will read your program and give points based on the points allocated to each component, the readability of your code (organization of the code and comments), logic, inclusion of the required functions, and correctness of the implementations of each function.

Rubric:

	Levels of Achievement						
Criteria	A	B	C	D	E	U	F
Specifications Weight 50.00%	100 % The program works and meets all of the specifications.	85 % The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	75 % The program produces mostly correct results but does not display them correctly and/or missing some specifications	65 % The program produces partially correct results, display problems and/or missing specifications	35 % Program compiles and runs and attempts specifications, but several problems exist	20 % Code does not compile and run. Produces excessive incorrect results	0 % Code does not compile. Barely an attempt was made at specifications.
Code Quality Weight 20.00%	100 % Code is written clearly	85 % Code readability is less	75 % The code is readable only by someone who knows what it is supposed to be doing.	65 % Code is using single letter variables, poorly organized	35 % The code is poorly organized and very difficult to read.	20 % Code uses excessive single letter identifiers. Excessively poorly organized.	0 % Code is incomprehensible
Documentation Weight 15.00%	100 % Code is very well commented	85 % Commenting is simple but solid	75 % Commenting is severely lacking	65 % Bare minimum commenting	35 % Comments are poor	20 % Only the header comment exists identifying the student.	0 % Non existent
Efficiency Weight 15.00%	100 % The code is extremely efficient without sacrificing readability and understanding.	85 % The code is fairly efficient without sacrificing readability and understanding.	75 % The code is brute force but concise.	65 % The code is brute force and unnecessarily long.	35 % The code is huge and appears to be patched together.	20 % The code has created very poor runtimes for much simpler faster algorithms.	0 % Code is incomprehensible

What to Submit?

You are required to submit your solutions in a compressed format (.zip). Zip all files into a single zip file. Make sure your compressed file is labeled correctly - <lastname>_<fistname>_Assn2.zip (e.g.: doe_john_Assn3.zip)

The compressed file MUST contain the following:

- <lastname>_<fistname>_assn2.c (or .cpp)
  - e.g.:
    - doe_john_assn2.c
    - doe_john_assn2.cpp

No other files should be in the compressed folder.

If multiple submissions are made, the most recent submission will be graded, even if the assignment is submitted late.

Where to Submit?

All submissions must be electronically submitted to the respected homework link in the course web page where you downloaded the assignment.

Academic Integrity and Honor Code.

You are encouraged to cooperate in study group on learning the course materials. However, you may not cooperate on preparing the individual assignments. Anything that you turn in must be your own work: You must write up your own solution with your own understanding. If you use an idea that is found in a book or from other sources, or that was developed by someone else or jointly with some group, make sure you acknowledge the source and/or the names of the persons in the write-up for each problem. When you help your peers, you should never show your work to them. All assignment questions must be asked in the course discussion board. Asking assignment questions or making your assignment available in the public websites before the assignment due will be considered cheating.

*The instructor and the TA will **CAREFULLY** check any possible proliferation or plagiarism. We will use the document/program comparison tools like MOSS (Measure Of Software Similarity: http://moss.stanford.edu/) to check any assignment that you submitted for grading. The Ira A. Fulton Schools of Engineering expect all students to adhere to ASU's policy on Academic Dishonesty. These policies can be found in the Code of Student Conduct:*

http://www.asu.edu/studentaffairs/studentlife/judicial/academic_integrity.htm

ALL cases of cheating or plagiarism will be handed to the Dean's office. Penalties include a failing grade in the class, a note on your official transcript that shows you were punished for cheating, suspension, expulsion and revocation of already awarded degrees.