Step 9: Strings and File I/O

This assigment is done in class on (monday)03-17-14 AND (friday)03-14-14.

This assignment is due for monday class 03-17-14 AND for the firday class 03-14-14.

This assignment is nearly all tutorial and should be completed during the class period.

Getting Started

Create a new program called stringer.c with the following code:

```
#include <stdio.h>

/*
 * Name : <Insert name>
 * Program to experiment with strings
 */

int main()
{

}
```

Be sure it compiles and runs. Of course, it won't do anything, yet.

1. More Strings and String Functions

Add this string variable to your main function:

```
    char word[] = "chudge";
```

This has created an array of characters (type char) that has 7 locations. The first 6 store the word "chudge". The 7'th is a zero. We can print the characters in this string with a loop. Add this code to your main function:

```
    for(i=0;  i<7;  i++)
    {
        printf("Location %d: %c\n", i, word[i]);
    }
```

Remember, an array in C starts with location zero (0). So, the seven locations are 0 to 6. Also, be sure you add any variables my code needs.

When you run this, the output should look like this:

```
Location 0: c
Location 1: h
Location 2: u
Location 3: d
Location 4: g
Location 5: e
Location 6:
```

Characters in C are just numbers stored in a single BYTE of memory. Change the printf statement to the following to see the actual numeric values of the characters:

```
        printf("Location %d: %d\n", i, word[i]);
```

Notice how the last location is a zero. That's the string null termination.

We can use that null termination to tell when the string ends. Replace the entire *for* loop with this code:

```
    for(i=0;  word[i] != '\0';  i++)
    {
        printf("Location %d: %c\n", i, word[i]);
    }
```

You should see the characters that make up the word, but not that location 6 this time. This is the reason for the null termination, so you can tell when the string ends.

I would like a way to know how long a string is. C provides a function, but let's write one ourselves first.

When writing a function, first answer these questions:

1. What is the input to the function?
2. What is the output of the function?
3. What does the function do?

We are writing a function that computes the length of a string. The input is a string we will test. The output is a number. Since there is only one output, we can make that number a return value. The function will iterate over the characters until it finds the null terminator, keeping a count of how many characters it finds.

First, let's create an empty version of the function. That's a common way to start. Here's an empty version. Add this to your program:

```
int StringLength(char str[])
{
    return 0;
}
```

The input is a string, which is an array of characters. So, my input parameter is an array of characters. The output of the function is an integer, the length of the string.

Add this function and a declaration for the function to your program and make sure it compiles okay.

Notice how I filled the function with *dummy code* that doesn't do anything other than return a value. When you write computer software, you want to write as little as possible, be sure it compiles, and get that working. Add this code to the main function to call this function:

```
    len = StringLength(word);
    printf("The string %s is %d characters long\n", word, len);
```

Be sure to define the variable len (of type int). This should run and output this:

```
The string chudge is 0 characters long
```

That is obviously wrong, but that's because we still have *dummy code* in the function. We've not filled it in, yet.

Okay, we want to count the number of characters in the string. We'll need a variable to do that. Add this variable to the function **StringLength**:

```
    int numChars = 0;
```

Note how I start with a value of zero. We're going to count letters. Think of this as your fingers. You just have none of them raised, yet.

It will take a loop to count the characters. The loop will continue until we get to the null termination. Each time we will test location numChars. If it's a character, we increment numChars. Here's the code to do this:

```
    while(str[numChars] != '\0')
    {
        numChars++;
    }
```

Be sure this compiles okay. It won't work, yet. Do you see why?

Now make the function return the number we compute:

```
    return numChars;
```

This should work and indicate that the word is 6 characters long.

Of course, the C library provides a function to do this as well. First, add this #include to the beginning of your program to indicate that you are using the string functions:

```
#include <string.h>
```

Now, change the line in main that calls StringLength() to this:

```
    len = strlen(word);
```

This should work fine.

Let's change the program and allow you to enter your own word. Add this new variable to main():

```
    char myWord[80];
```

Now add this code to input a word:

```
    printf("Enter a word: ");
    scanf("%s", myWord);
    printf("The entered word is: %s\n", myWord);
```

This should run and output your word.

Try typing this input: my word

You can see that scanf inputs one word. It stops when it gets to a space or the end of a line. It's a good way to enter one word at a time.

The scanf function used like this is very dangerous. If you input a word longer than 80 characters, it will keep writing past the end of the array. If you enter a really long word, you may crash the program or get garbage in the output or, worst of all, see nothing wrong, even though your program is now broken. Try that. I just held down a key and let it repeat characters until I knew I had well more than 80..

Using scanf like this is one of the most common security holes in C programs, since it lets a user write beyond the end of an array using bad input. They can write things into the memory of your computer and change the way a program executes.

For that reason, you should always tell it the maximum length word you can handle in the array. Since we have to *leave space for the null terminator*, that's 79 characters in this case. We can tell scanf to stop at 79 characters this way:

```
    scanf("%79s", myWord);
```

This should work the same. But, if you enter a word 79 characters or longer, it will write until then and stop. So, we need to be sure there is a null termination on the string. I added this line of code to fix that right after the scanf statement:

```
    myWord[79] = '\0';
```

This should work and input words up to 79 characters long. Anything longer will be ignored.

At this point, delete the code for that loop we created earlier that outputs the characters in the string and add this code to the end of the main function to output the length of the entered word:

```
    printf("The string %s is %d characters long\n", myWord, strlen(myWord));
```

This should output the length of your entered word.

Your program should look like this at this point:

```
#include <stdio.h>
#include <string.h>

/*
 * Name : <Insert name>
 * Program to experiment with strings
 */

int StringLength(char str[]);

int main ()
{
    char word[] = "chudge";
    char myWord[80];
    int i;
    int len;

    printf("Enter a word: ");
    scanf("%79s", myWord);
    myWord[79] = '\0';
    printf("The entered word is: %s\n", myWord);


    len = strlen(word);
    printf("The string %s is %d characters long\n", word, len);
    printf("The string %s is %d characters long\n", myWord, strlen(myWord));

}

int StringLength(char str[])
{
    int numChars = 0;

    while(str[numChars] != '\0')
    {
        numChars++;
    }

    return numChars;
}
```

Notice that I have **repeated code**:

```
    len = strlen(word);
    printf("The string %s is %d characters long\n", word, len);
    printf("The string %s is %d characters long\n", myWord, strlen(myWord));
```

These lines of code do the same thing twice on two different string variables. This is a perfect place to use a function. Add this function that does the same thing as these lines:

```
void PrintLength(char str[])
{
    printf("The string %s is %d characters long\n", str, strlen(str));
}
```

Then replace those redundant lines of code with these function calls:

```
    PrintLength(word);
    PrintLength(myWord);
```

This should run exactly the same.

To be sure, the program at this point looks like:

```
#include <stdio.h>
#include <string.h>

/*
 * Name : <Insert name>
 * Program to experiment with strings
 */

int StringLength(char str[]);
void PrintLength(char str[]);

int main ()
{
    char word[] = "chudge";
    char myWord[80];
    int i;
    int len;

    printf("Enter a word: ");
    scanf("%79s", myWord);
    myWord[79] = '\0';
    printf("The entered word is: %s\n", myWord);

    PrintLength(word);
    PrintLength(myWord);
}

void PrintLength(char str[])
{
    printf("The string %s is %d characters long\n", str, strlen(str));
}


int StringLength(char str[])
{
    int numChars = 0;

    while(str[numChars] != '\0')
    {
        numChars++;
    }

    return numChars;
}
```

Suppose we want to enter a sentence rather than a string? To do so we use fgets. Add this new variable to put our sentence in:

```
    char mySentence[80];
```

Then add the following code to input the sentence using fgets. Add this before the code that enters the word.

```
    printf("Enter a sentence: ");
    fgets(mySentence, 80, stdin);
    printf("The entered sentence is: %s\n", mySentence);
```

This should work and the output look something like this:

```
cbowen@ubuntu:~/cse251$ ./stringer
Enter a sentence: This is a test
The entered sentence is: This is a test

Enter a word: word
The entered word is: word
The string chudge is 6 characters long
The string word is 4 characters long
```

You may have noted that extra blank line. The reason for that is that fgets gets the entire sentence up to and include the newline character where you hit return. Add this code to output the length of the sentence:

```
    PrintLength(mySentence);
```

Then try entering the sentence: This is a test

The output should look something like this:

```
cbowen@ubuntu:~/cse251$ ./stringer
Enter a sentence: This is a test
The entered sentence is: This is a test

Enter a word: word
The entered word is: word
The string chudge is 6 characters long
The string word is 4 characters long
The string This is a test
 is 15 characters long
```

See how it says there are 15 characters? But, if you count, you will count 14 in "This is a test". The extra character is a newline character. Suppose we don't want that newline character? Well, we can ask how long the string is using strlen:

```
    len = strlen(mySentence);
```

In this case, len is 15. Locations 0 to 13 are the 14 characters of "This is a test". Location 14 is the newline character. We can get rid of it by adding this line of code:

```
    mySentence[len - 1] = '\0';
```

The way this should look is:

```
    printf("Enter a sentence: ");
    fgets(mySentence, 80, stdin);
    len = strlen(mySentence);
    mySentence[len - 1] = '\0';
    printf("The entered sentence is: %s\n", mySentence);
```

This should work as you would expect it to.

The following is optional and should be done only if you have time after finishing Section 1:

I asked you to put the code that enters the sentence before the code that enters the word. Try reversing them like this:

```
    printf("Enter a word: ");
    scanf("%79s", myWord);
    myWord[79] = '\0';
    printf("The entered word is: %s\n", myWord);

    printf("Enter a sentence: ");
    fgets(mySentence, 80, stdin);
    len = strlen(mySentence);
    mySentence[len - 1] = '\0';
    printf("The entered sentence is: %s\n", mySentence);
```

When you run your program it is going to do something strange. See if you can figure out why and a way to fix it.

2. String Functions

I'm only going to work with sentences in this section. So, delete the variables: word and myWord, the code the inputs myWord, and the code that outputs the length of word and myWord. After deletions your code should look something like this:

```
#include <stdio.h>
#include <string.h>

/*
 * Name : <Insert name>
 * Program to experiment with strings
 */

int StringLength(char str[]);
void PrintLength(char str[]);

int main ()
{
    char mySentence[80];
    int len;

    printf("Enter a sentence: ");
    fgets(mySentence, 80, stdin);
    len = strlen(mySentence);
    mySentence[len - 1] = '\0';
    printf("The entered sentence is: %s\n", mySentence);

    PrintLength(mySentence);
}

void PrintLength(char str[])
{
    printf("The string %s is %d characters long\n", str, strlen(str));
}
```

```
int StringLength(char str[])
{
    int numChars = 0;

    while(str[numChars] != '\0')
    {
        numChars++;
    }

    return numChars;
}
```

Before you do anything, make sure this is working.

First, let's try that reverse function. Add the Reverse function to your program:

```
void Reverse(char str[] )
{
    int front = 0;
    int back = strlen(str) - 1;
    char t;             /* A temporary place to put a character */

    while (front < back)
    {
        t = str[front];
        str[front] = str[back];
        str[back] = t;
        front++;
        back--;
    }
}
```

Don't forget the declaration. Be sure this compiles.

Now add this code to the end of main to test the reverse function:

```
    Reverse(mySentence);
    printf("%s\n", mySentence);
```

Test this to be sure it is working.

Have you read that code to try to figure out what it does? Often I want to know what is going on inside code. To do so, add printf statement that tell you what is happening. For example, I might add this line to Reverse to help me understand what front and back do:

```
void Reverse(char str[] )
{
    int front = 0;
    int back = strlen(str) - 1;
    char t;             /* A temporary place to put a character */

    while (front < back)
    {
        printf("front=%d back=%d\n", front, back);
        t = str[front];
        str[front] = str[back];
        str[back] = t;
        front++;
        back--;
    }
}
```

Try this and see if this helps you understand what the function is doing and how it works. When done, remove the line.

Are you sure you understand what this code is doing:

```
        t = str[front];
        str[front] = str[back];
        str[back] = t;
```

In this case "t" is a temporary place to keep a character. If you don't understand this, consider adding printf statements to help you see what is happening.

I want to write a function to tell me how many spaces are in a sentence. The input is a character string and the output is a number. So, add this empty function as a start for a function to do this:

```
int NumberOfSpaces(char str[])
{
    return 0;
}
```

I'll not put run icons in this section. I'll leave that to you.

Again, I started with an empty function. I then added code to call this function to main:

```
    printf("The number of spaces is: %d\n", NumberOfSpaces(mySentence));
```

I do suggest you add this before you reverse the string, BTW.

So, the function needs to iterate over every character in the string. We did this before, remember? Here's a loop we did earlier. Don't add this, yet.

```
    for(i=0;  word[i] != '\0';  i++)
    {
        printf("Location %d: %c\n", i, word[i]);
    }
```

Of course, in our function, the variable is str, not word. So, make that change and add this version to NumberOfSpaces:

```
    for(i=0;  str[i] != '\0';  i++)
    {
        printf("Location %d: %c\n", i, str[i]);
    }
```

Don't forget to declare i. When you run this it should output the characters. Note how I remembered something I did before and copied that code, modifying it to suit my function. Of course, it does the wrong thing right now. We want to count the space. So, add a variable to count with:

```
    int numSpaces = 0;
```

Now delete that printf statement in the loop body. Instead, just test to see if we have a space. If so, increment our counter:

```
        if(str[i] == ' ')
            numSpaces++;
```

Change your function to return numSpaces and it should completely work. Just to be sure, NumberOfSpaces should look like this:

```
int NumberOfSpaces(char str[])
{
    int i;
    int numSpaces = 0;

    for(i=0;  str[i] != '\0';  i++)
    {
        if(str[i] == ' ')
            numSpaces++;
    }

    return numSpaces;
}
```

Turn in stringer.c via http://secure.cse.msu.edu/handin/

If you have time, try creating a function that will count any possible character:

```
int NumberOfAppearances(char str[], char ch)
{
}
```

Write this function yourself and try to make it case-insensitive. For example, NumberOfAppearances("I really like that I can do this", 'i'); should return 4. Consider using the character functions from slide 26.

3. File I/O

Create a new program reverser.c that initially looks like this:

```
#include <stdio.h>
#include <string.h>

/*
 * Name : <Insert name>
 * Program to experiment with files
 */

 int main(int argc, char *argv[])
 {
    if(argc < 3)
    {
        printf("Insufficient arguments\n");
        return 1;
    }

    printf("Copy from file: %s\n", argv[1]);
    printf("Copy to file: %s\n", argv[2]);

 }
```

Be sure this compiles okay. When you run it, use this command:

```
./reverser reverser.c result.txt
```

This program uses arguments to the main function. C passes an array of strings to your program if you include the following parameters on the main function:

```
int main(int argc, char *argv[])
```

The first values is the number of arguments. I provided three above. The first is always the program name as invoked: ./reverser. You can add this line to check that if you want:

```
    printf("%s\n", argv[0]);
```

The second is reverser.c. The third is result.txt. This is a way to write a program you can pass information to when you run it. It's a common way to write utilities or programs that work on files.

Add that Reverse function from stringer.c to this program. We're going to use it in this program as well.

When we work with a file in C, we have a file "handle". It's a way to keep track of what file we are working on. Add these two variables that are file handles to the main function:

```
    FILE *inFile;
    FILE *outFile;
```

Be sure you use all upper case for FILE. Be sure this compiles okay.

Let's open the input file first. Add this line of code to the end of main to open the input file:

```
   inFile = fopen(argv[1], "r");
```

Opening a file means we have made a connection to the file and are now able to access it. Because I used "r" in the fopen call, we are *reading* this file.

Always check to ensure an open worked okay. If the files does not exist, we don't want to proceed. Here's code for that check:

```
    if(inFile == NULL)
    {
        printf("Unable to open file %s\n", argv[1]);
        return 1;
    }
```

Be sure to try this with both a good and bad file name as the first argument on the command line. For example, try:

```
./reverser reverser.c result.txt
```

This should work okay. Then try:

```
./reverser aardvark.dat result.txt
```

It should say:

```
cbowen@ubuntu:~/cse251a$ ./reverser aardvark.dat result.txt
Copy from file: aardvark.dat
Copy to file: result.txt
Unable to open file aardvark.dat
```

We need a loop to read all of the lines of text from the input file. First, add a variable to put a line of text into:

```
    char line[120];
```

Then add this loop that reads the file and prints it to the terminal to the end of main:

```
    /* While we are not at the end of the file */
    while(!feof(inFile))
    {
        /* Read a line of text from the file */
        fgets(line, sizeof(line), inFile);

        /* Remove the newline at the end of the line */
        if(strlen(line) > 0)
            line[strlen(line) - 1] = '\0';

        printf("%s\n", line);
    }
```

This example introduces a new C features: sizeof(). You use sizeof like it is a function. But, it's actually built into C and is a compiler feature. It returns an integer that is the size of the item in Bytes. Note that this is not the number of items in an array. It is for a character array, since the size of each element is one byte. But, if you want to know the size of an integer array in elements, you have to do: sizeof(intarray) / sizeof(int). This only works in the function where the array has been declared and is not always usable. The compiler will complain if it's not available at that point. It's great for fgets, though.

Be sure this runs. It should output your program if you run it like this:

```
./reverser reverser.c result.txt
```

So, what does this do? The loop continues until the end of the file. We get a line from the file using fgets. I remove the newline at the end of each line using line[strlen(line) - 1] = '\0'; But, only do this if the length of the input is greater than zero. The fgets function may return a zero length string at the end of the file. Hence the test to be sure line is more than 0 characters long.

The last thing we always do is to close the file. Add this line at the end to close the input file:

```
   fclose(inFile);
```

Should work and output whatever file you are reading from.

Now add the following code to open the output file. Add this before the loop that reads the file:

```
    outFile = fopen(argv[2], "w");
    if(outFile == NULL)
    {
        printf("Unable to open file %s\n", argv[2]);
        return 1;
    }
```

If you run this, it should create the output file. It will be empty, but should show up in your directory if you do an **ls** command.

Be sure to close the file at the end of the program:

```
   fclose(outFile);
```

Change this line:

```
        printf("%s\n", line);
```

to:

```
        fprintf(outFile, "%s\n", line);
```

When you run this program, it will copy your program to the file result.txt. Look at that file to be sure it is working.

To be sure, your program should look like this at this point:

```
#include <stdio.h>
#include <string.h>

/*
 * Name : <Insert name>
 * Program to experiment with files
 */


void Reverse(char str[] );


int main(int argc, char *argv[])
{
    FILE *inFile;
    FILE *outFile;
    char line[120];

    if(argc < 3)
    {
        printf("Insufficient arguments\n");
        return 1;
    }

    printf("Copy from file: %s\n", argv[1]);
    printf("Copy to file: %s\n", argv[2]);

    inFile = fopen(argv[1], "r");
    if(inFile == NULL)
    {
        printf("Unable to open file %s\n", argv[1]);
        return 1;
    }

    outFile = fopen(argv[2], "w");
    if(outFile == NULL)
    {
        printf("Unable to open file %s\n", argv[2]);
        return 1;
    }

    /* While we are not at the end of the file */
    while(!feof(inFile))
    {
        /* Read a line of text from the file */
        fgets(line, sizeof(line), inFile);

        /* Remove the newline at the end of the line */
        if(strlen(line) > 0)
            line[strlen(line) - 1] = '\0';

        fprintf(outFile, "%s\n", line);
    }

    fclose(inFile);
    fclose(outFile);
 }



void Reverse(char str[] )
{
    int front = 0;
    int back = strlen(str) - 1;
    char t;             /* A temporary place to put a character */

    while (front < back)
    {
        t = str[front];
        str[front] = str[back];
        str[back] = t;
        front++;
        back--;
    }
}
```

Now, make your program reverse the characters in every line of the file it writes. As an example, the first few lines of result.txt should look like this:

```
>h.oidts< edulcni#
>h.gnirts< edulcni#

 */
>eman tresnI< : emaN *
selif htiw tnemirepxe ot margorP *
/*
```

Create a file named: backwards.txt and put this text into it:

```
reverof kees nac uoy ,ksid raelc a nO
```

Turn in reverser.c via http://secure.cse.msu.edu/handin/

Final Tasks

There are no new tasks at the end of this assignment. Please proceed to Project 2 immediately.

And we are done.

CSE 251