**CSE 278 Computer Architecture**

**Homework #2**

|  |  |
| --- | --- |
| **Name:** | Samuel Mallamaci, Joey Seiler, Jess Fielder |

### Part 1: Crash course in C

* Answer the following questions
* Try to identify the key to each problem and keep your answers concise and to the point; 2-3 sentences should suffice.
* These questions bring up important points about pointer usage and control flow in C.  Keep these in mind when working on the remainder of the homework.

1. Consider the following C program.

#include <string.h>

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

{

char \*temp;

strcpy(temp, argv[0]);

return 0;

}

Why is the above code incorrect (i.e., likely to crash)?

|  |
| --- |
| ‘temp’ has been created but it is not pointing to a location that the string can be copied to. ‘temp’ should have been initialized like ‘char \*temp = (char \*)malloc(20);’ |

1. Consider the following C program.

#include <string.h>

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

{

char temp[9];

strcpy(temp, argv[0]);

return 0;

}

A buffer overflow occurs when the program name is 9 characters long (e.g., "12345.exe"). Why?

|  |
| --- |
| Strings require a termination character like ‘\0’; there is not enough space for ‘12345.exe\0’ in a 9 character array. |

1. Consider the following C program.

#include <string.h>

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

{

char \*buffer = "Hello";

strcpy(buffer, "World");

return 0;

}

Why does this program crash?

|  |
| --- |
| The program crashes because you can’t initialize a char pointer to a string literal in c. |

1. Consider the following C snippet.

void myfunc()

{

char b[100];

char \*buffer = &b[0];

strcpy(buffer, "World");

}

Is this correct? What's a simpler expression for &b[0]?

|  |
| --- |
| This works, but a simpler expression would have been ‘char \*buffer = b;’ because the array is accessed as a pointer to its first element. |

1. Consider the following C program.

#include <stdio.h>

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

{

printf("%s %s %s\n",\*argv, (\*(argv+1)) + 2, \*(argv+2));

return 0;

}

If this code is executed using the following line, what will be the output?

> program1 -n5 abc

|  |
| --- |
| program1 5 abc |

1. Consider the following C program.

#include <stdio.h>

#include <string.h>

char \*myfunc(char \*\*argv)

{

char buffer[100];

strcpy(buffer, "hello");

return buffer;

}

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

{

char \*s = myfunc(argv);

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

}

What's wrong with this?

|  |
| --- |
| This is printing the actual pointer, not what the pointer is referring to. |

### 

### Part 2: Fixing the Bugs

Examine the provided program main.c. The purpose of this program is to count words specified as command-line arguments.  Read the description of the program and its functionality in the comment at the top of main.c.  Now read through the rest of main.c and the Makefile and understand what each part does.

Finally, compile and run the program from the shell:

> make

(ignore the compiler warning for now)

> ./main

The program compiles and links... so it must work!  But is it really doing what it is supposed to do?

**Answer the following questions and fix the corresponding bugs in main.c. Again, try to keep your answers brief and focused.**

### Explain why this program uses the exclamation operator with the strcmp() function.

|  |
| --- |
| If the strings that strcmp() is comparing are the same, then it returns 0. However, when this is inside an if-statement, the 0 stops the code from executing. The program needs the code to execute when the strings are the same, so the ! operator is used with the strcmp() function. |

### Explain why the 'LENGTH' macro returns the length of an array. Would it work with a pointer to a dynamically allocated array? (Hint: understand sizeof).

|  |
| --- |
| Sizeof returns the number of bytes in an array, so it divides it by the size of the type of the array. It would not work with a dynamically allocated array because sizeof() returns the number of bytes allocated to an element.  CAUTION: sizeof applied to a pointer variable that points to an array (dynamically allocated or otherwise) gives the size of the pointer, not the size of the array! i.e.:  int a[10];  int \*b = a;  /\* sizeof (b) != sizeof (a) \*/ |

### Explain and fix the logical flow bug within the switch statement. (What happens when the -h option is used?)

|  |
| --- |
| The –h flag is supposed to print the help information for the program.  Fixed by adding break statements. |

### Explain and fix the argument parsing error. (Why is entrycount never zero?)

|  |
| --- |
| Entrycount is always going to be less than LENGTH(entries), so it will always be incremented once. Since it is always incremented, it will never be equal to zero in the if statement. |

### Fix print\_result() to print results correctly and in the same order as the words were specified on the command line. Explain your solution.

|  |
| --- |
| We looped stating at 1 and increasing while 'i' was less than 'entry\_count' so that './main' wasn't printed. We accessed the values inside 'entries' correctly, based on what value we were at in the loop. |

### Part 3: Enhancements

Now that the bugs have been ironed out, it's time to add some functionality to our word counting program.  Follow the instructions below to complete the word counter.

Add the following features to the program:

1. Alter the program such that only the correct output is sent to the standard output stream (stdout), while error and help messages are sent to the standard error stream (stderr).  (Hint: use fprintf.)

See the expected output listed in the comment at the top of main.c for an example of what should go to stdout.

1. Implement an optional command-line switch '-fFILENAME' that sends program output to a file named FILENAME (i.e., filename specified as a command line argument).
2. Add support for matching arbitrary numbers of words, not just 5. (Hint: Use malloc. It's ok if you allocate a bit more memory than is actually used.
3. Safeguard the program from buffer overflow attacks, in which data is written beyond the end of a memory allocation. (Hint 1: ‘gets’ is BAD.  Use fgets instead, which specifies the maximum number of characters to be read in. Hint 2: Be careful about the newline character '\n' at the end of the line; gets and fgets handle it differently.)
4. Allow multiple words to be specified per line. Words may be separated by spaces or by any punctuation, including slashes and quotation marks. (Hint 1: Understand strtok. Hint 2: Recall how to escape special characters in strings.)

## 