**Operating Systems**

**Homework #1-Part B**

|  |  |
| --- | --- |
| **Name:** | Yan Yu |

**Part B: Review of C**

### Sub 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)?

|  |
| --- |
| Since we want to copy the string pointed by argv[0], and we declared a char pointer temp, but we never allocated memory space to the temp pointer, so strcpy will cause memory access violation error. |

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?

|  |
| --- |
| The string size pointer by argv[0] is larger than the char array capacity. |

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?

|  |
| --- |
| We declared a char pointer buffer point to a string literal (constant), and we are trying to modify the string literal pointed by the pointer, which is undefined behavior and will crash the program. |

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]?

|  |
| --- |
| The program is right, b has enough space to copy “World”; a simpler expression for &b[0] is just b itself. |

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?

|  |
| --- |
| Inside myfunc, buffer is a local variable which means that after myfunc return, the memory associated with buffer will be deallocated automatically by OS, so the information will lost. When we try to print the result, we will get error output. |

### 

### Sub 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.

|  |
| --- |
| In C, since there is no Boolean type, so often function will return an integer to indicate some status information, the strcmp will return 0 if two strings are equal and non-zero integer to indicate that the two strings are not equal. So, using exclamation operator is just a way to indicate that condition is true, since !0 will be a non-zero value, which C thinks it is true in a condition check. |

### 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).

|  |
| --- |
| Since the LENGTH macro is just a wrapper of sizeof(s) / sizeof(\*s), so when we apply  LENGTH to an array of any type, LENGTH will be replaced by the actual expression. And sizeof(s) will return the total byte size of the actual array, sizeof(\*s) will return the byte size of the actual type the array stores, so the expression will return the actual length of the array we provide.  And for dynamically allocated array, length won’t work, since s will be a pointer type not the actual array type, so sizeof(s) will probably return 4 or 8 depends on the OS and implementation of C, and sizeof(\*s) will return the byte size of the first element in the dynamic array, which is the byte size of the actual stored type. So, LENGTH behavior is undefined in this context, depend on what the array stores.  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?)

|  |
| --- |
| When I typed ./main –h command in the terminal, the default action of the switch statement is executed, which should not happen, instead the program should print the help message, this can be fixed by increae the pointer argv value by 1. And also, a break statement is needed at the end if case ‘h’, in order to prevent the fall through |

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

|  |
| --- |
| When parsing the arguments, the first argv should be ignored, since it points to the program name, so before while loop, the argv value should be increased by one, this will prevent entryCount never zero. |

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

|  |
| --- |
| Inside print result function, entries array never proceeds to the next value, so the result will print the first word in argument passed into the program multiple times. |

### Sub 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.)

## 