(CSCI 363) Sample Machine Project Zero: Counting Words

Purpose:

This project is designed for students to review C programming language. It helps the students create, debug and extend C programs that run within a shell environment and utilize basic I/O and string manipulation functions.

For those students, whose Computer Science I class didn't use C programming language, this is also a catch up project. Students have four weeks to review/learn C beside learning the new materials from textbook. It should be enough as long as the students truly learned from Computer Science I and Computer Science II since all programming languages have the similar syntax and logic structure.

Turn in:

Turn in two files only: main.c and YourNameProj0.docx, which is a word file that contains the answers to the questions listed in this file below.
Due date will be announced on Blackboard.

The scenario:

your well-intentioned-but-inexperienced pair-programming buddy has just written some code for the first assignment. Unfortunately, they dropped the course/were abducted by aliens, and it is now up to you to pick up where they left off.

The accompanying program files (Makefile and main.c) contain examples of good and bad programming practices and includes deliberate errors. Your job is to find and fix the errors, implement missing features, and learn some tricks of the trade in the process.

Luckily, a test program is included to help you along. You can (and should) use it to check your work. See the Self-Evaluation section for details.

There are three parts to this Project.

Part One: Crash Course in C

Read the Review section at the end of the file. The list of topics it contains is useful for self-assessment and as a study guide.

Answer the questions in Pointers in C section below. 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 in C. Keep these in mind when working on the remainder of the Project.

Part Two: Fixing the Bugs

The purpose of the provided 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 (Refer to the file "How to Run C programs from Command Line.pdf"):

```
> 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 questions in Section Debugging and fix the corresponding bugs in main.c. Again, try to keep your answers brief and focused.

Part Three: Enhancements

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

Self-Evaluation

The testrunner program is included so that you can check your progress as you implement different parts of the Project.

To run all tests, type the following in command line:

```
> make test
```

To run a specific test, e.g., stderr_output, type the following in command line:

```
> ./main -test stderr_output
```

As you can see, testrunner is implemented entirely in C. If you found this Project too easy, or are just plain curious, feel free to look at the implementation and see if you can figure out how everything works. The relevant files are: smp0_test.* and testrunner.*.

Note: You should remove or disable any additional debugging output you may have created before running the tests. One way to do this easily is through the use of the preprocessor directive #ifdef:

```
#ifdef DEBUG
fprintf(stderr, "My string %s %d\n", var1, var2);
#endif
```

Then add -DDEBUG to the CCOPTS line in the Makefile during development, and remove it before testing. Make sure that your code still compiles and runs without the debug output!

Question and Coding Sections

Pointers in C

Students must turn in the answers to these questions to get 25% credit of Project 0.

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)?

2) 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?

3) 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?

4) 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]?

5) 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
```

6) 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?

<u>Fixing the Bugs</u>

Students must turn in the answers to these questions to get 25% credit of Project 0.

Understanding the code

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

2) 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).

Bug hunting

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

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

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

<u>Enhancements</u>

Student must code these functionalities to get 50% of Project 0 credit. 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.

2) 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).

3) Add support for matching arbitrary numbers of words, not just 5. (Hint: use malloc, and don't worry too much about memory efficiency).

4) Safeguard the program from buffer overflow attacks. (Hint: 'gets' is BAD. Use fgets instead, which specifies the maximum number of characters to be read in.)

5) Allow multiple words to be specified per line. (Hint 1: Understand 'strtok'. Hint 2: Be careful about the newline character '\n' at the end of the line.)

Review of C

Project0 is a crash course in C. These are some of the topics you will need to keep in mind for all of the projects through the semester.

Common gotchas for new C programmers:
- C strings
- switch, case, break statements
- pointers
- pointers to pointers
- pointer arithmetic
- using argv and argc
- ?: syntax
- comma operator (only used occasionally, but can be useful)
- static, heap and stack variables
- C library functions for string handling

Pointers and dereferencing syntax (* &)

String handling and library I/O:
- strcmp, strcpy, strlen, strcat
- atoi, atol
- puts, fputs, printf, fprintf, fopen, fclose, scanf
- using man pages to determine which library must be included

Low-level system I/O:
- open, write, close

Concepts:
- declaration vs. definition of variables and functions
- scope of functions and variables
- how the linker glues separate compilation units together
- header files and preprocessor directives (#include, #ifdef)

Skills:
- using man pages to look up information, e.g., man strcmp
- understanding compiler and runtime messages
- using gdb to debug crashes (segfaults)
- using an editor to write code

- creating and modifying make files
- divide and conquer approaches to bugs and implementation problems

Defensive programming and debugging:
- using assert.h
- using write, fprintf(stderr, format, ...)

- Read the tutorials and take the quizzes here:
http://www.cprogramming.com/tutorial.html#ctutorial

Man pages:
When you need specific information about some C function (e.g., return values, argument types, multithreading safety properties), man pages may provide the answer. Run "man <function name>" from the shell to view the appropriate man page.