CSCD 437 Lab 4
Team 6
Ryan Cranston, Petal Michaud, Julian Welge

Problem a)

**a.**

```
int foo(char *arg, char *out)
{
  strcpy(out, arg);
  return 0;
}

int main(int argc, char *argv[])
{
  char buf[64];
  if (argc != 2)
  {
    fprintf(stderr, "a: argc != 2\n");
    exit(EXIT_FAILURE);
  }// end if

  foo(argv[1], buf);

  return 0;

}// end main
```

Sample compile before adjustment:

```
julian@julian-VirtualBox:~/Desktop/lab4$ gcc problem1.c
problem1.c: In function 'foo':
problem1.c:3:1: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]
    3 | strcpy(out, arg);
      | ^~~~~~
problem1.c:3:1: warning: incompatible implicit declaration of built-in function 'strcpy'
problem1.c:1:1: note: include '<string.h>' or provide a declaration of 'strcpy'
  +++ |+#include <string.h>
    1 | int foo(char *arg, char *out)
problem1.c: In function 'main':
problem1.c:11:1: warning: implicit declaration of function 'fprintf' [-Wimplicit-function-declaration]
   11 | fprintf(stderr, "a: argc != 2\n");
      | ^~~~~~~
problem1.c:11:1: warning: incompatible implicit declaration of built-in function 'fprintf'
problem1.c:1:1: note: include '<stdio.h>' or provide a declaration of 'fprintf'
  +++ |+#include <stdio.h>
    1 | int foo(char *arg, char *out)
problem1.c:11:9: error: 'stderr' undeclared (first use in this function)
   11 | fprintf(stderr, "a: argc != 2\n");
      |         ^~~~~~
problem1.c:11:9: note: 'stderr' is defined in header '<stdio.h>'; did you forget to '#include <stdio.h>'?
problem1.c:11:9: note: each undeclared identifier is reported only once for each function it appears in
problem1.c:12:1: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
   12 | exit(EXIT_FAILURE);
      | ^~~~
problem1.c:12:1: warning: incompatible implicit declaration of built-in function 'exit'
problem1.c:1:1: note: include '<stdlib.h>' or provide a declaration of 'exit'
  +++ |+#include <stdlib.h>
    1 | int foo(char *arg, char *out)
problem1.c:12:6: error: 'EXIT_FAILURE' undeclared (first use in this function)
   12 | exit(EXIT_FAILURE);
      |      ^~~~~~~~~~~~
julian@julian-VirtualBox:~/Desktop/lab4$
```

First problem that jumps out from our class discussions is the use of strcpy(). One solution is to use strncopy() and then include passing the size of the buffer into the foo() function to use for strncopy(). Also have to put in the #include <string.h>, <stdio.h>, and <stdlib.h>. Something like the following:

```
 GNU nano 4.8                                    problem1.c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int foo(char *arg, char *out, int bufSize)
{
strncpy(out, arg, bufSize);
return 0;
}
int main(int argc, char *argv[])
{
char buf[64];
if (argc != 2)
{
fprintf(stderr, "a: argc != 2\n");
exit(EXIT_FAILURE);
}// end if
foo(argv[1], buf, sizeof(buf));
return 0;
}// end main
```

Here is the output to the above code:

```
julian@julian-VirtualBox:~/Desktop/lab4$ gcc -o problem1 problem1.c
julian@julian-VirtualBox:~/Desktop/lab4$ ./problem1 shesellsseashellsdownbytheseashorewithsunnywhoisgladitissunny
julian@julian-VirtualBox:~/Desktop/lab4$
```

No warnings or errors ☺

---

Problem b)

**b.**

```
int foo(char *arg)
{

  char buf[128];

  int len, i;


  len = strlen(arg);

  if (len > 136)

    len = 136;


  for (i = 0; i <= len; i++)

    buf[i] = arg[i];

  return 0;

}

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

{

  if (argc != 2)

  {

    fprintf(stderr, "b: argc != 2\n");

    exit(EXIT_FAILURE);

  }

  foo(argv[1]);

  return 0;

}// end main
```

Besides the #include statements, one piece that stands out is that the max length of the len and the buffer are different so those can be made equal  the for loop should stop at an index less than the length and so you could make sure your string is null terminating.

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

int foo(char *arg)
{
      char buf[128];
      strncpy(buf, arg, sizeof(buf));
      return 0;
}
int main(int argc, char *argv[])
{
      if (argc != 2)
      {
            fprintf(stderr, "b: argc != 2\n");
            exit(EXIT_FAILURE);
      }
      foo(argv[1]);
      return 0;
}// end main
```

```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./b1.out AccordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyItswingsaretoosmalltogetitsfatli
ttlebodyoffthegroundThebeeofcourseflies
```

A solution to this problem would be to cut out the problem code and use strncpy() and limit the copying size to buf and including a \0, a problem caused by this solution is the input string gets cut off once it runs out of memory.

**Code 2: "b.c"**

**Code 3: "c.c"**

```
int bar(char *arg, char *targ, int ltarg)
{
      int len, i;len = strlen(arg);
      if (len > ltarg)
            len = ltarg;
      for (i = 0; i <= len; i++)
            targ[i] = arg[i];
      return 0;
}
int foo(char *arg)
{
      char buf[128];
      bar(arg, buf, 140);
      return 0;
}
int main(int argc, char *argv[])
{
      if (argc != 2)
```

```
      {
            fprintf(stderr, "c: argc != 2\n");
            exit(EXIT_FAILURE);
      }
      foo(argv[1]);
      return 0;
}// end main
```

The issue with the code above the receiving string of buf is not big enough to hold what is being put into it by the bar function so we end up with a segmentation fault.

```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./c.out AccordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyItswingsaretoosmalltogetitsfatlit
tlebodyoffthegroundThebeeofcourseflies
zsh: segmentation fault  ./c.out
```

Similar to the last problem, the solution would be to avoid the overflow by making the sizes the same between the input and the buffer. We also don't want to use strlen() because the input could be bigger than what it says. We can use a loop instead to figure the size.

**Code 4: "d.c"**

```
int foo(char *arg, short arglen)
{
      char buf[1024];
      int i, maxlen = 1024;
      if (arglen < maxlen)
      {
            for (i = 0; i < strlen(arg); i++)
                buf[i] = arg[i];
      }
      return 0;
}
int main(int argc, char *argv[])
{
      if (argc != 2)
      {
            fprintf(stderr, "d: argc != 2\n");
            exit(EXIT_FAILURE);
      }
      foo(argv[1], strlen(argv[1]));
      return 0;
}
```

Most noticeable thing for use the #include statements and that the for loop should use the short arglen instead of finding the strlen(arg) again.