CS 491: Homework 1, Test Case Report

Armando Diaz Tolentino (adt), adiazt2@uic.edu

February 13, 2012

Abstract

For this assignment, I have re-implemented the standard C libraries for memory and string handling. Functions like memcpy(), memset(), strcat(), and strcmp(). For more information refer to the README file and doxygen documentation provided.

1 Provided Test Cases

The test cases provided in the assignment handout are given in this section:

1.1 Case 1

Test Case :

```
void main(int argc, char **argv) {

char* cat = "cat" ;
char* dogGoesWoof ="dog\0goes\0woof" ;
char str[100] ;
memset( str, 0, 100 ) ;
strcat( str, cat ) ;
printf( "%s", str) ; // test 1
printf("\n") ;
strcpy( str, dogGoesWoof ) ;
printf( "%s", str) ; // test 2
printf("\n") ;
strcpy( str+1, str ) ;
memcpy( str, cat, 22 ) ;
printf( "%s", str) ; // test  3
}
```

As expected, str prints correctly for the first two cases. For the third case, we get a segmentation fault. This occurs because of the command "strcpy(str+1, str) ; " My implementation of strcpy(), like the standard library implementations, fails in the case where the two input strings overlap in memory.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
cat
dog
Segmentation fault
```

1.2 Case 2

```
void main(int argc, char **argv) {
char* ptr = 0 ;
*ptr = 'a' ;
}
```

As expected a pointer to NULL, yields a run-time error. This would actually have nothing to do with my code.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
Segmentation fault
```

2 Additional Test Cases

2.1 More Args than Format Specifiers

Here we test for giving printf() more arguments than the format string specifies.

```
void main(int argc, char **argv) {

char* name = "Ra's Al Ghul", *hero = "Batman" ;
int someNum = 19872, otherNum = 45 ;
long lNum = 998292 ;

/* more arguments than format specifiers */
printf( "%s\n", name, hero, someNum, otherNum, lNum ) ;
printf( "%d\n", someNum, hero, name, otherNum, lNum ) ;
printf( "%ld\n", lNum, hero) ;
}
```

As expected additional arguments are simply not printed. No error occurs.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
Ra's Al Ghul
19872
998292
```

2.2 More Format Specifiers than Args: Missing String

Here we test behavior when more format flags are provided, than arguments to printf().

```
void main(int argc, char **argv) {
char* hello = "Konnichiwa, minasan :)" ;
printf("%s\n%s\n%s\n") ;
}
```

In the case where a string is expected first, and no arguments are given, a NULL value is used for the string. This caused my ASSERT macro to be called, when printf() attempted to copy the non-existent string to the memory buffer provided.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst

Assert failed: dest != NULL && src != NULL
 in file mem.c
 at line 109

Terminated
```

This second case demonstrates what occurs when a string argument is missing, but another at least has been provided.

```
void main(int argc, char **argv) {
char* hello = "Konnichiwa, minasan :)" ;
printf("%s\n%s\n%s\n", hello ) ;
}
```

In this case garbage values are printed out. We begin printing from a certain location in memory, and continue until an NULL terminator is encountered. I am not certain why in particular this happens. It is my understanding that the va_arg() function returns a 0 value, when no further arguments have been provided. In this case, we should get an ASSERT failure as in the previous test case.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
Konnichiwa, minasan :)
t,T$
       e3
PDD
```

2.3 More Format Specifiers than Args: Missing int

Here we test the output of printf() when an integer value is missing.

```
void main(int argc, char **argv) {
char* hello = "Konnichiwa, minasan :)" ;
printf("%s\n%d\n%d\n", hello, 4544) ;
}
```

In the place of the given integer flag, printf() prints a garbage value for the integer. Two test cases below demonstrate that the integer isn't the same every time we run our code.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
Konnichiwa, minasan :)
4544
-216081116


cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
Konnichiwa, minasan :)
4544
-217113308
```

2.4 Parameter Type Mismatch : string as an int

Here we test printf() behavior when a string is interpreted as an integer type.

```
void main(int argc, char **argv) {
char* iSay="Chunky Bacon ~===~ !!!!!!!" ;
printf( "%x\n", iSay) ;
}
```

printf() ends up printing the memory address that the string begins at. This is not surprising considering basic knowledge of c-strings.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
0x804955C
```

2.5 Parameter Type Mismatch : int as a string

Here we test what occurs when an integer is interpreted as a string pointer.

```
void main(int argc, char **argv) {
printf( "%s\n", 2343432) ;
}
```

As expected an arbitrary int is interpreted as a memory address which we don't in this case have access to. Thus a segmentation fault occurs.

```
cer0@skynet:~/Documents/courses/cs491/projects/hw1$ ./tst
Segmentation fault
```

3 Conclusion

The above represents only a sampling of tests done to verify the contents of hw1. The other library functions provided were also extensively tested, but their test cases are not provided here for brevity's sake.