ECS 30 **Homework #6 (50 points)** **Fall 2013**

Due: Wednesday, November 13th, Written: 4pm in 2131 Kemper. Program: 11:59pm using handin to cs30, p6 directory. Filenames: reverse.c, suffix.c, names.c

Written (8 points) pp. 511-512 1, 2, 3, 4, 5, 8, 9, 10.

"Refer to these declarations when determining when determining the effect of the statements in Questions 1-4.

```
char s5[5], s10[10], s20[20];
char aday[7] = "Sunday";
char another[9] = "Saturday";
```

```
1. strncpy(s5, another, 4);  s5[4] = '\0';
```

```
2. strcpy(s10, &aday[3]);
```

```
3. strlen(another);
```

```
4. strcpy(s20, aday); strcat(s20, another);
```

5. Write a function that pads a variable-length string with blanks to its maximum size. For example, if `s10` is a ten-character array currently holding the string "screen", **`blank_pad`** would add three blanks (one of which would overwrite the null character) and finish the string with the null character. Be sure your function would work if no blank padding were necessary.

8. Which one of the following would call somefun only if the string values of character arrays a and b were equal?

```
a. if (strcmp(a, b))
     somefun();

b. if(strcmp(a, b) == 0)
     somefun();
```

```
c. if(a == b)
     somefun();

d. if(a[] == b[])
     somefun();
```

9. What does this program fragment display?

```
char x[80] = "gorilla";
char y[80] = "giraffe";
strcpy(x, y);
printf("%s %s\n", x, y);
```

```
a. gorilla giraffe
b. giraffegorilla gorilla
```

```
c. gorilla gorilla
d. giraffe giraffe
```

10. What does this program fragment display?

```
char x[80] = "gorilla";
char y[80] = "giraffe";
strcat(x, y);
printf("%s %s\n", x, y);
```

```
a. gorillagiraffe giraffe
b. giraffegorilla gorilla
```

```
c. gorilla gorilla
d. giraffe giraffe
```

Programming (42 points)

All programs should be able to compile with no warnings when compiled with the –Wall option. Remember to add the –g option on all of your gcc lines so that you can use gdb to debug your programs. You should put your name(s) in a comment on the first line of each file. The prompts, and output format of each program must match the examples exactly. main() should be the first function in each file. You may assume the user will enter valid values.

1. p. 515 #7 (10 points, 10 minutes) Filename: reverse.c

"Write a program that takes a data line at a time and reverses the words of the line. For examples,

```
Input: birds and bees
Reversed: bees and birds
```

The data should have one blank between each pair of words."

Additional Specifications:

Since gets() is unsafe, and causes a warning, you should use fgets with `stdin` as the FILE* to read the entire line. `stdin` is defined in stdio.h as a FILE* that is the keyboard stream. I used strtok(), sprintf(), strcpy() and a temp array to iteratively create the reversed string. The first time you call strtok() with a given string, s, you use s as its first parameter, e.g. `char *ptr = strtok(s, "aY,");` For all subsequent calls to strtok to parse the same string, you provide NULL as its first parameter, e.g. `ptr = strtok(NULL, "\nk");` Note that the second parameter is the a list of delimiter characters, any one of which will be used to separate the tokens. Consecutive delimiters in the string being parsed will be treated as one delimiter, e.g. "SomeYaaYwords" will return a pointer to "words" when the second strtok is called because the "YaaY" will have been marked with '\0''s after the first call. As demonstrated, the delimiter string need not be the same for each call to strtok(). In the case of reverse.c, the delimiters are space and newline characters. You may assume no line will be longer than 79 characters.

```
[ssdavis@lect1 p6]$ reverse.out
Input: birds and bees
Reversed: bees and birds
[ssdavis@lect1 p6]$ reverse.out
Input: Now is the time for all good men to come to the aid of their country
Reversed: country their of aid the to come to men good all for time the is Now
[ssdavis@lect1 p6]$
```

2. p. 515 #8 (5 points, 5 minutes) Filename: suffix.c

"Write and test a function that finds and returns through an output parameter the longest common suffix of two words (e.g. the longest common suffix of "procrastination" and "destination" is "stination," of "globally" and "internally" is "ally," and of "gloves" and "dove" is the empty string)."

Additional Specifications:

main() should read and write the words. Hint: start at the end of the words, and use two char*.

3. pp. 515-516 #9 (30 minutes, 25 points) Filename: names.c

"Write a program that processes a data file of names in which each name is on a separate line of at most 80 characters. Here are two sample names:

```
Hartman-Montgomery, Jane R.
Doe, J. D.
```

On each line the surname is followed by a comma and a space. next comes the first name or initial, then a space and the middle initial. Your program should scan the names into three arrays—`surname, first, and middle_init`. If the surname is longer than 15 characters, store only the first 15. Similarly, limit the first name to 10 characters. Do not store periods in the first and middle_init arrays. Write the array's contents to a file, aligning the contents of each column:

```
Hartman-Mongom  Jane            R
Doe             J               D
```

Additional specifications:

The names of the two files will be passed as command line parameters. The first command line will be the input filename. The first line of the file will have an int that says how many names are in the file. main() will use this number to dynamically allocate all three arrays using malloc. `surname` and `first`, like argv, will be char**'s because each will hold the address of an array of char*. You will need to use a for loop to create char arrays of size 16 to be stored in those two char** arrays. Since each element of `middle_init` element will only be a single char, it can be a char* of the size specified on the first line of the file.

After reading all of the names in read_names(), your program should call sort() to sort the three parallel arrays at the same time based on the surname. After calling sort(), main() will call write_names() which will write the names to the file named in the second command line parameter using the format specified. The function called by main(), free_memory() will deallocate all of the dynamically allocated memory.

Extra credit (5 points): Have your sort function take into account the first names of people who have identical surnames.

```
[ssdavis@lect1 p6]$ cat names.txt
6
Ryan, Elizabeth O.
McIntyre, O. J.
Cauble-Chantrenne, Kristin K.
Larson, Lois F.
Thorpe, Trinity R.
Ruiz, Pedro M.
[ssdavis@lect1 p6]$ names.out names.txt names.res
[ssdavis@lect1 p6]$ cat names.res
Cauble-Chantren Kristin         K
Larson          Lois            F
McIntyre        O               J
Ruiz            Pedro           M
Ryan            Elizabeth       O
Thorpe          Trinity         R
[ssdavis@lect1 p6]$ cat names2.txt
9
Clinton, Hillary R.
Bonds, Bobby S.
Bonds, Barry L.
Clinton, William I.
Clinton, Chelsea T.
Bush, Laura M.
Bush, George W.
Bush, Jenna F.
Bush, Barbara G.
[ssdavis@lect1 p6]$ names.out names2.txt names2.res
[ssdavis@lect1 p6]$ cat names2.res
Bonds           Barry           L
Bonds           Bobby           S
Bush            Barbara         G
Bush            George          W
Bush            Jenna           F
Bush            Laura           M
Clinton         Chelsea         T
Clinton         Hillary         R
Clinton         William         I
[ssdavis@lect1 p6]$
```