Project[8]: Command Line Arguments/Dynamic Memory Allocation/String library

Due Tuesday, 12/05/2017,11:59am

Project Goals

The goals of this project are to:

1. Get you familiar with command line arguments and dynamic memory allocation.
2. Get you familiar with functions to read from and write to files, in text and binary modes.

Important Notes:

1. **Formatting:** Make sure that you follow the precise recommendations for the output content and formatting. Your assignment will be auto-graded and any change in formatting will result in a loss in the grade.
2. **Comments:** Header comments are required on all files and recommended for the rest of the program. Points will be deducted if no header comments are included.

Problem 1 (Combining strings):

Write a program that asks the user to enter two strings (of length at most 30) and then performs one of two possible operations provided as command line arguments to the program. The possible command line options and the associated operations are as follows:

- Option: "-i" Operation: if given this option, the program should create a new string that is built by interspersing the two strings, by alternatively placing one character at a time from each string. When one of the strings exhausted, the program should copy the remaining of the characters from the second string. This functionality should be implemented in a function called `intersperse`, which takes as parameters the two strings and returns a pointer to the newly created string. Your function should use dynamic memory allocation to generate the new string.

For example, if string 1 is "abcde" and string 2 is "1234567", the resulting string should be "a1b2c3d4e567".

- Option: "-w" Operation: if given this option, the program should create a new string that is build by concatenating the two given strings, in which there has been a '*' character inserted in between every character in those strings. There should be a '*' character between the two strings, but no such character at the end of string 2. This functionality should be implemented in a function called `widen_stars`, which takes as parameters the two strings and returns a pointer to the newly created string. Your function should use dynamic memory allocation to generate the new string.

For example, if string 1 is "abcde" and string 2 is "1234567", the resulting string should be "a*b*c*d*e*1*2*3*4*5*6*7".

Here is an example of exactly how the program should behave (items underlined are entered by the user):

```
> combine_strings -i
> Please enter a string of maximum 30 characters: abcde
> Please enter a string of maximum 30 characters: 1234567
> The combined string is: a1b2c3d4e567
```

or

```
> combine_strings -w
> Please enter a string of maximum 30 characters: abcde
> Please enter a string of maximum 30 characters: 1234567
> The combined string is: a*b*c*d*e*1*2*3*4*5*6*7
```

Additional constraints:

- Your program should use the exact form and spacing of the user prompts and output as indicated above
- Before terminating, your program should de-allocate the space allocated for the newly created strings.

Code for this program should be in a file named `combine_strings.c`

Problem 2

You will write a program that uses a pre-defined array of strings (the `planets` array discussed in class) and sorts the names of the planets in either alphabetical or reverse alphabetical order. The program will use command line arguments to read in the order in which the strings should be printed: ‟`-o  a`″ for alphabetical and ‟`-o r`″ for reverse alphabetical order. Your program should follow the constraints below:

- Implement a function `string_compare`, which takes as parameters two strings (`str1` and `str2`) and returns 0 if `str1` is prior to `str2` alphabetically, 1 if `str2` is prior to `str1` alphabetically and -1 if `str1` is identical to `str2`. The function should compare the two strings one character at a time and return the appropriate value as soon as a difference is noticed between the strings.

- Use the above function to “sort” the names of the planets by using the following strategy:

```
    for i = 0 to size of planets array - 1

    for j = size of planets array - 1 down to i + 1

    if planets[j] is prior to planets[j -1] alphabetically

  then exchange planets[j] with planets[j-1]
```

The exchange in the last line above will exchange the pointers stored in `planets[j]` and `planets[j-1]`. For strings that are identical (assuming that the names of the planets would be changed), there should be no exchange needed.

Note: the algorithm above sorts in alphabetical order. For reverse alphabetical order, the "if" test should be change accordingly.

Your program should function as follows:

```
./planets –o a
The planets in alphabetical order are: Earth, Jupiter, Mars, Mercury,
Neptune, Pluto, Saturn, Uranus, Venus
```

Alternatively, for reverse alphabetical order:

```
./planets –o r
The planets in reverse alphabetical order are: Venus, Uranus, Saturn,
Pluto, Neptune, Mercury, Mars, Jupiter, Earth
```

Your program should be saved in a file called `planets_sort.c`.

Grading Rubric

Grading will be done for each problem as follows:

Correctly-named file	5%
Header comment	2%
Program compiles	5%
Correctly-reading of command line arguments	28%
Follow specifications	30%
Correct result printed	30%

Submission details

To submit your project, you will have to save your project files to nomachine:

- create a directory called "project8"
- save your *.c files in that directory
- TO Submit:
  > cd project8
  > submit

The submission script copies all files in the current directory to our directory. You may submit as many times as you like before the deadline, we only keep the last submission.

Academic Honesty

Academic dishonesty is against university as well as the system community standards. Academic dishonesty includes, but is not limited to, the following:

Plagiarism: defined as submitting the language, ideas, thoughts or work of another as one's own; or assisting in the act of plagiarism by allowing one's work to be used in this fashion.

Cheating: defined as (1) obtaining or providing unauthorized information during an examination through verbal, visual or unauthorized use of books, notes, text and other materials; (2) obtaining or providing information concerning all or part of an examination prior to that examination; (3) taking an examination for another student, or arranging for another person to take an exam in one's place; (4) altering or changing test answers after submittal for grading, grades after grades have been awarded, or other academic records once these are official.

Cheating, plagiarism or otherwise obtaining grades under false pretenses" constitute academic dishonesty according to the code of this university. Academic dishonesty will not be tolerated and penalties can include canceling a student's enrollment without a grade, giving an F for the course, or for the assignment. For more details, see the University of Nevada, Reno General Catalog.