COL100 Lab 10

I semester 2016-17

October 3, 2016

Objective

To be able to write recursive programs in C.

Instructions

1. After 1 hour 45 minutes have passed, your code will be checked. Whatever you have completed till this point will be recorded. Anything that you complete later will not be recorded.

2. If you complete an assignment later, you can ask the TAs of your lab session any problems and doubts that you face. There is no need to show the TA your code, if there is no problem in it.

3. You cannot attend any lab session other that your allotted session, without informing the TAs of the session you are attending. This too is permitted only for genuine reasons.

4. Also, you will not get attendance, if you do not attend your own lab session, nor will your performance be noted. (Even if you fill in the attendance sheet, it will not be uploaded later.)

Programs

- Press Ctrl + Alt + T to open a terminal.
- cd to the directory COL100.
- In this directory, create another folder, called as lab10.
- cd to lab10.

1. Suppose we have bunnies standing in a line, numbered 1, 2, 3,... The odd bunnies (1, 3, ..) have 2 ears. The even bunnies (2, 4, ..) have 3 ears.Write a recursive program to compute and return the total number of ears of the bunnies standing in the line (without using loops or multiplication, use recursion).

   Input: Number of bunnies in a line, followed by a newline.
   Output: Total number of ears, followed by a newline.

2. Write a program to check whether the given string contains the correct pair of parenthesis. Your program should return 'true' if the string contains nesting of zero or more pairs of parenthesis, like "(())" or "((()))" otherwise return 'false'.

   Input: String of parenthesis, followed by a newline.
   Output: Return true if the string contains zero or more pairs of parenthesis else return false, followed by a newline.

3. Write a recursive program to compress a string. Your program should ask the user to enter a string. The compression technique should work as follows:
   If the same character in the string occurs continuously a multiple number of times then replace the multiple occurrences of that character by the single occurrence followed by the number of times it has occurred. If a character occurs exactly one number of time then represent it as it is.
   Examples:

   ```
   aabbbbaaccc - a2b4a2c3
   abccd - abc2d
   ucds - ucds
   ```

   Input: a string, followed by a newline.
   Output: a compressed string, followed by a newline.

4. Write a program to compute the super-digit of a given integer. Super-digit of an integer is the one digit number which is the sum of the digits in that number.
   For example, super-digit of 9875 will be calculated as:

   ```
   super-digit(9875) = super-digit(9+8+7+5)
                     = super-digit(29)
   ```

```
= super-digit(2+9)
= super-digit(11)
= super-digit(1+1)
= super-digit(2)
= 2
```

Input: an integer, followed by a newline.
Output: a super-digit number, followed by a newline.

Optional Programs

1. Write a program for a given number n that will print all primes smaller than or equal to n.
   For example:
   1) If n is 10, the output should be 2, 3, 5, 7.
   2) If n is 20, the output should be 2, 3, 5, 7, 11, 13, 17, 19.

   The sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than 10 million or so. Assume that, initially every number between 2 to n, is marked true i.e. a prime number.
   Using the following algorithm find all the prime numbers less than or equal to a given integer n by Eratosthenes' method and unmark the rest of the number:

   (a) Create a list of consecutive integers from 2 to n: (2, 3, 4, ..., n).
   (b) Initially, let p equal 2, the first prime number.
   (c) Starting from p+1 to n, unmark all the multiples of p because those numbers are not the prime numbers.
   (d) Now, find the next number greater than p in the list that is marked. If there was no such number, stop. Otherwise, let p now equal this number (which is the next prime), and repeat from step (c).
   When the algorithm terminates, all the numbers in the list that are marked are prime.

   Input: an integer n, followed by a newline.
   Output: all prime numbers each in newline, smaller than or equal to n, followed by a newline.

2. Write a recursive program to find the difference between the sum of the squares of the first n natural numbers and the square of their sum. Your program should ask the user to enter the value of n.
   Example: The sum of the squares of the first ten natural numbers is, $1^2 + 2^2 + ... + 10^2 = 385$
   The square of the sum of the first ten natural numbers is, $(1 + 2 + ... + 10)^2 = 55^2 = 3025$ Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 - 385 = 2640.

Useful Commands in Linux

1. Open terminal: Ctrl + Alt + T
2. Terminate current Linux command: Ctrl + C
3. Make a new directory: mkdir *dirname*
4. Copy: cp *src dest*
5. Rename: mv *originalname newname*
6. Delete: rm *filename*
7. Change working directory: cd *path*
8. List contents of a folder: ls
9. List contents of a folder including hidden files: ls -a
10. Print current directory: pwd

Points to Remember

1. To set proxy: Open an internet browser and set the Automatic proxy configuration url to http://www.cc.iitd.ernet.in/cgi-bin/proxy.btech (or proxy.dual if you are a Dual Degree student).
   (For Firefox, open Options > Advanced > Network Tab > (Connection) Settings > Choose "Automatic proxy configuration" and set the URL)

Optional : Use vim editor

1. Open a file: vim filename.txt
2. Insert in a file: i (insert mode) (Use Esc to come out of the insert mode)
3. Navigation: arrow keys
4. Undo u
5. Redo Ctrl+R
6. Saving a file :w
7. Closing a file without saving :q!
8. Saving and closing a file :wq
9. Deleting a line dd
10. Copying a line yy
11. Pasting a line p