Informe - Trabajo práctico N°1

DC – Paste – More & Strutils

75.41 Algoritmos y Programación II

Cátedra Wachenchauzer

Cuatrimestre II, 2016

Matías Scakosky - 38707339 - Legajo: 99627

**Ayudante individual: Constanza González**

Objetivos:

* Lograr utilizar los TDA implementados durante el cuatrimestre para una utilidad en particular que no son únicamente pruebas unitarias.
* Lograr un mejor manejo de la consola de LINUX y su interacción con C mediante argv[] y argc.
* Utilizar lo aprendido en el cuatrimestre y aplicarlo para lograr un mejor manejo de cadenas.
* Utilizar el manejo de cadenas para las aplicaciones para poder probarlas y testearlas a medida que se creaban las utilidades y aplicaciones
* Familiarizarse con los archivos de que C abre cuando se ejecuta un programa y poder diferenciarlos. Como STDIN, STDOUT y STDERR

1. Etapa diseño y resoluciones lógicas.  
El TP1 está dividido en dos grandes partes, la primera son aplicaciones de uso por consola referidas al manejo de archivos, la segunda utilidades de manejo de cadenas las cuales serán analizadas de manera separada y en detalle cada una de ellas con el objetivo de entender cuáles fueron las primeras vicisitudes de cada una de las utilidades propuestas por la catedra

1.1 Manejo de archivos y utilidades de LINUX

Cada una de las utilidades está conformada por un diseño distinto debido al comportamiento diverso de cada una de las mismas.

* Paste  
  Esta utilidad debe recibir dos archivos de texto y poner uno al lado de otro y en cuyo caso que uno de los dos tenga más líneas que otro se debe ejecutar el programa hasta donde haya coincidencia en la cantidad de líneas y lanzar un mensaje de error informando al usuario sobre el problema.  
  Para ello, el algoritmo a implementar deberá leer línea a línea cada uno de los dos archivos y de manera instantánea y local ir imprimiendo las líneas de cada una de los dos archivos hasta llegar al final de alguno de los dos archivos y saber si el otro tiene más líneas. Para lograr colocar una línea de un archivo junto a la otra, el algoritmo deberá reconocer el salto de línea del primero de los archivos de texto que este leyendo para transformarlo en una tabulación respecto de la segunda línea del siguiente archivo.  
  Luego de la idea básica de Paste, las siguientes preguntas surgieron en la etapa de diseño. ¿Qué pasa cuando el archivo llega al final? Es evidente que si la última línea del primer archivo termina o no con un salto de línea, será distinto el comportamiento ¿Cómo lograr que el comportamiento sea igual para ambos casos? Si la última línea del segundo archivo no contiene un fin de línea o un salto de línea ¿Cómo hacer para agregarlo y no modificar la estética de Paste?
* More  
  Esta utilidad debe recibir un archivo de texto un numero n entero y debe imprimir las primeras n líneas del archivo dejando que el usuario imprima las próximas m-n líneas (siendo m el número total de líneas del archivo) ingresando un salto de línea.  
  El algoritmo a implementar deberá recorrer las primeras n líneas e imprimirlas una a una. Luego deberá esperar a que el usuario decida ingresar la cadena correspondiente para que, por cada una de las mismas el archivo el prograna imprima una linea más. En cuyo caso que n sea mayor o igual a la cantidad de líneas que tiene el archivo, el mismo programa imprimirá todas las líneas y no finalizara, debido a que no hay mas líneas que imprimir.  
  Para More las siguientes preguntas surgieron en la etapa de diseño. ¿Habra algun problema si cuando al final de las n líneas pasadas por parámetro, la linea n en su último caracter no contiene un salto de linea?
* DC  
  Esta utilidad debe recibir por entrada estándar una cadena de números y operaciones matemáticas en formato de calculadora polaca inversa y resolver la operación pedida (siempre y cuando la misma tenga solución válida). Para ello el algoritmo a implementar deberá ir apilando los números que lleguen por entrada estándar y cuando el programa detecte una operación matemática desapilará los primeros dos números, concretará la operación y luego el resultado de la misma lo volverá a apilar.  
  Para la implementación de este algoritmo hay varias consideraciones a tener en cuenta. Por ejemplo, ¿Cómo separar las cosas ingresadas por la entrada estándar de manera eficiente para tener claro que es lo que hay que calcular? ¿Qué pasa si lo ingresado no es válido? ¿Si hay más operaciones que números en la pila? ¿Si lo ingresado no son números ni operaciones? ¿Si se divide por 0?

1.2 Manejo de cadenas

* Split()  
  La función Split() divide una cadena en sub cadenas en cada ocurrencia de un caracter de separación determinado. Para ello el algoritmo a implementar debe reconocer en la cadena de caracteres cuantos separadores hay para poder crear un arreglo de cadenas donde en cada una de las posiciones guarde toda las sub cadenas hasta el final. Para ello debe poder recorrer la cadena y recordar donde estaba el ultimo separador para luego ir guardando desde el ultimo separador hasta el siguiente y en cuyo caso que no haya caracteres en el medio debe guardar una cadena vacía, la última posición del arreglo debe ser para todas las cadenas el mismo, debe ser NULL.  
   Para Split(), en su etapa de diseño fueron surgiendo las siguientes preguntas. ¿Cuánta memoria tengo que pedir según la cadena? ¿Cómo marco la cadena de manera tal que cuando quiera guardar la siguiente sub cadena arrancar desde la posición siguiente sin modificar la cadena original?
* Join()  
  La función Join() es la inversa del Split, recibe un arreglo de sub cadenas y dado un separador debe devolver una única cadena donde cada una de esas subcadenas estén separadas por el separador indicado. La complejidad algoritimica de Join() debe ser O(n) siendo n el largo de la cadena resultante, para ello el algoritmo a implementar debe primero, medir el largo total de la cadena resultante contando la cantidad de caracteres de cada una de las sub cadenas, y luego agregarle la cantidad de separadores correspondiente, luego de haber hecho eso se pedirá memoria para todo ese bloque resultante y luego se llenara cada una de las posiciones de la memoria pedida con los caracteres de todo el arreglo que llega por parámetro.   
  Join(), en su diseño presentó las siguientes incertezas, ¿Qué pasa con la memoria si el arreglo que llega esta vacio, o si contiene una única sub cadena asociada y la cadena esta vacia? ¿Cuánta memoria debo pedir para un arreglo de subcadenas vacias? ¿Cómo lograr el orden de complejidad pedido?
* Free\_strv()  
  La función Free\_strv() libera la memoria asociada con un arreglo dinámico de cadenas dinámicas, no presentó mayores dificultades a la hora de su diseño

2. Implementación y complejidad algorítmica de la resolución