Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática
CI-1201 Programación II - 2012b
Profesor Jeisson Hidalgo-Céspedes
17-Setiembre-2012. Duración 2 horas

Examen 01

En cada ejercicio se evaluará la eficiencia del código, el uso de identificadores significativos, la indentación, escritura correcta de llaves {} y el uso adecuado de la palabra reservada const. Se dispone de dos horas para entregar la prueba y debe realizarse en forma estrictamente individual.

Implemente en C o C++ un comando `sar` (*search and replace*) que ayude al usuario a encontrar o reemplazar segmentos de texto en archivos de texto o la entrada estándar. Su programa deberá comportarse como un comando normal de Unix. Para efectos de este examen, asuma que el usuario siempre lo invocará con la siguiente sintaxis:

```
$ sar -i -W search-text -r replacement-text FILES
```

El parámetro `-i` indica que la búsqueda debe ignorar mayúsculas y minúsculas. El parámetro `-W` indica que se debe sobrescribir los archivos de entrada. El parámetro `search-text` es el texto a buscar, el cual debe ser reemplazado por el texto `replacement-text` (antecedido por la opción `-r`) en todos los archivos `FILES`.

En general su comando debe hacer lo siguiente. Abrir cada uno de los archivos indicados por la lista FILES, y procesarlos línea a línea hasta encontrar el fin de archivo. En cada línea se debe buscar ignorando mayúsculas y minúsculas el *texto de búsqueda* y si se encuentra, reemplazarlo por el *texto de reemplazo*. Sea que se haya o no reemplazado el *texto de búsqueda*, la línea debe escribirse en el archivo destino.

Por efecto de la opción `-W`, su programa debe abrir cada archivo por parámetro y el resultado escribirlo en un archivo temporal. Puede utilizar el mismo nombre del archivo original concatenándole la extensión `.tmp` ó `.sar`. Para efectos de este enunciado asuma que este archivo no existe, y si existiese, será truncado sin aviso. Una vez terminado el procesamiento, el archivo original debe ser eliminado con la función `remove()`, y el archivo temporal ser renombrado al original con la función `rename()`, ambas de la biblioteca `<stdio.h>`.

Evaluación

1. [20%] Procesa y reacciona adecuadamente a los parámetros de invocación. Puede asumir que el usuario siempre provee los parámetros en el mismo orden.
2. [30%] Su programa lee líneas de cada archivo listado en FILES en orden, reemplaza el texto de búsqueda por el texto de reemplazo y las guarda en archivos temporales que luego reemplazan a los originales. Si hay algún error con el manejo de archivos (no existen, no hay espacio en disco, etc.) se reportan en el error estándar. Cierra archivos tan pronto como se dejen de utilizar.
3. [15%] Hace un adecuado uso de la memoria. Trabaja a nivel de línea. Si utiliza `fgets()` maneja adecuadamente cambios de línea al final de las cadenas devueltas. Si utiliza memoria dinámica no provoca fugas de memoria.
4. [35%] La función de búsqueda de texto hace la menor cantidad de recorridos posibles. Ignora mayúsculas y minúsculas. Hace el reemplazo correctamente en los archivos destino. Debe ser una función libre o un método de clase.
5. [10% **Opcional**] Haga que su programa pueda leer archivos que contengan líneas de cualquier longitud. Para esto debe llamar repetidamente a `fgets()` incrementando el tamaño del *buffer* hasta que encuentre el final de línea.

Material de referencia

- `int tolower( int ch );`
  Retorna el resultado de convertir el carácter `ch` a minúsculas, contrario a `toupper(ch)`. Declarada en `<ctype.h>`.

- `FILE * fopen( const char * filename, const char * mode );`
  Abre el archivo cuyo nombre es indicado por `filename` utilizando uno de los modos: `"r"` (sólo lectura, el archivo debe exisitr), `"w"` (escritura, será sobrescrito con uno vacío si existe), `"a"` (añadir al final, se creará si no existe). Se debe agregar una `"b"` al modo si se quiere trabajar con arcivos binarios. Retorna un puntero hacia una estructura que representa el archivo. Declarada en `<stdio.h>`.

- `int fclose( FILE * stream );`
  Cierra el archivo. Retorna 0 en caso de éxito, `EOF` en caso de fallo. Declarada en `<stdio.h>`.

- `int remove( const char * filename );`
  Elimina el archivo cuyo nombre es dado por parámetro. Retorna 0 en caso de éxito. Declarada en `<stdio.h>`.

- `int rename( const char * oldname, const char * newname );`
  Renombra el archivo o directorio con nombre `oldname` a `newname`. Si el archivo destino existe, podría ser sobrescrito. Retorna 0 en caso de éxito. Declarada en `<stdio.h>`.

- `char * fgets( char * str, int num, FILE * stream );`
  Lee caracteres de `stream` y los aloja en `str` hasta encontrar un cambio de línea (`'\n'`) (el cual es incluido en `str`) o llenar la capacidad máxima del `str` indicada por `num` caracteres; lo que ocurra primero. Retorna `str` en caso de éxito, o `NULL` en caso de fallo. Declarada en `<stdio.h>`.

- `int fputs( const char * str, FILE * stream );`
  Escribe los caracteres de la cadena `str` en el archivo `stream` hasta encontrar el final de cadena (`'\0'`). Retorna `EOF` en caso de fallo. Declarada en `<stdio.h>`.

- `int fprintf( FILE * stream, const char * format, ... );`
  Imprime la cadena con formato `format` en el archivo `stream`. Por cada especificador `%` se espera un parámetro opcional. Algunos especificadores son: `%c` (carácter), `%s` (cadena de caracteres terminada en `'\0'`), `%i` (entero con signo), `%u` (entero sin signo), `%f` (flotante con decimales), `%%` (el carácter `%`). Retorna la cantidad de caracteres escritas en el archivo en caso de éxito, o un número negativo en caso de fallo. Declarada en `<stdio.h>`.

- `char * strcpy( char * destination, const char * source );`
  Copia todos los caracteres de la cadena `source` en `destination` incluyendo el fin de cadena (`'\0'`). Retorna `destination`. Declarada en `<string.h>`.

- `char * strcat( char * destination, const char * source );`
  Agrega `source` al final de `destination`. Retorna `destination`. Declarada en `<string.h>`.