:]

hagan lo suyo cerebritos

Boberias varias sobre Git:

**git add lab1/parsing.c** (agrega tu archivo a un “area de preparacion/espera)

>si queres sacar un archivo de esta area de preparacion:

**git restore --staged lab1/parsing.c** (no modifica tu archivo, solo lo saca del waiting list)

**git status**

**git commit -m "..."**

**git push**

checkear commits

**git log** (apretar q para salir)

memory leaks: punteros/TADS q usan punteros/funciones con malloc (que no son los que devuelve la funcion) que quedan sin liberar

Command.c:

scommand comando simple struct

implementacion como lista

*// ([char\*],char\*,char\*)*

struct scommand\_s{

GSList\* cmd\_list;

char\* input\_redirection;

char\* output\_redirection;

};

lista cmd\_list

Command.h:

*/\* scommand: comando simple.*

*\* Ejemplo: ls -l ej1.c > out < in*

*\* Se presenta como una secuencia de cadenas donde la primera se denomina*

*\* comando y desde la segunda se denominan argumentos.*

*\* Almacena dos cadenas que representan los redirectores de entrada y salida.*

*\* Cualquiera de ellos puede estar NULL indicando que no hay redirección.*

*\**

*\* En general, todas las operaciones hacen que el TAD adquiera propiedad de*

*\* los argumentos que le pasan. Es decir, el llamador queda desligado de la*

*\* memoria utilizada, y el TAD se encarga de liberarla.*

*\**

*\* Externamente se presenta como una secuencia de strings donde:*

*\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\* front -> | cmd | arg1 | arg2 | ... | argn | <-back*

*\* ---------------------------------*

*\**

*\* La interfaz es esencialmente la de una cola. A eso se le*

*\* agrega dos accesores/modificadores para redirección de entrada y salida.*

*\*/*

typedef struct scommand\_s \* scommand;

scommand scommand\_new(void);

*/\**

*\* Nuevo `scommand', sin comandos o argumentos y los redirectores vacíos*

*\* Returns: nuevo comando simple sin ninguna cadena y redirectores vacíos.*

*\* Ensures: result != NULL && scommand\_is\_empty (result) &&*

*\* scommand\_get\_redir\_in (result) == NULL &&*

*\* scommand\_get\_redir\_out (result) == NULL*

*\*/*

scommand scommand\_destroy(scommand *self*);

*/\**

*\* Destruye `self'.*

*\* self: comando simple a destruir.*

*\* Requires: self != NULL*

*\* Ensures: result == NULL*

*\*/*

*/\* Modificadores \*/*

void scommand\_push\_back(scommand *self*, char \* *argument*);

*/\**

*\* Agrega por detrás una cadena a la secuencia de cadenas.*

*\* self: comando simple al cual agregarle la cadena.*

*\* argument: cadena a agregar. El TAD se apropia de la referencia.*

*\* Requires: self!=NULL && argument!=NULL*

*\* Ensures: !scommand\_is\_empty()*

*\*/*

void scommand\_pop\_front(scommand *self*);

*/\**

*\* Quita la cadena de adelante de la secuencia de cadenas.*

*\* self: comando simple al cual sacarle la cadena del frente.*

*\* Requires: self!=NULL && !scommand\_is\_empty(self)*

*\*/*

void scommand\_set\_redir\_in(scommand *self*, char \* *filename*);

void scommand\_set\_redir\_out(scommand *self*, char \* *filename*);

*/\**

*\* Define la redirección de entrada (salida).*

*\* self: comando simple al cual establecer la redirección de entrada (salida).*

*\* filename: cadena con el nombre del archivo de la redirección*

*\* o NULL si no se quiere redirección. El TAD se apropia de la referencia.*

*\* Requires: self!=NULL*

*\*/*

*/\* Proyectores \*/*

bool scommand\_is\_empty(const scommand *self*);

*/\**

*\* Indica si la secuencia de cadenas tiene longitud 0.*

*\* self: comando simple a decidir si está vacío.*

*\* Returns: ¿Está vacío de cadenas el comando simple?*

*\* Requires: self!=NULL*

*\*/*

unsigned int scommand\_length(const scommand *self*);

*/\**

*\* Da la longitud de la secuencia cadenas que contiene el comando simple.*

*\* self: comando simple a medir.*

*\* Returns: largo del comando simple.*

*\* Requires: self!=NULL*

*\* Ensures: (scommand\_length(self)==0) == scommand\_is\_empty()*

*\**

*\*/*

char \* scommand\_front(const scommand *self*);

*/\**

*\* Toma la cadena de adelante de la secuencia de cadenas.*

*\* self: comando simple al cual tomarle la cadena del frente.*

*\* Returns: cadena del frente. La cadena retornada sigue siendo propiedad*

*\* del TAD, y debería considerarse inválida si luego se llaman a*

*\* modificadores del TAD. Hacer una copia si se necesita una cadena propia.*

*\* Requires: self!=NULL && !scommand\_is\_empty(self)*

*\* Ensures: result!=NULL*

*\*/*

char \* scommand\_get\_redir\_in(const scommand *self*);

char \* scommand\_get\_redir\_out(const scommand *self*);

*/\**

*\* Obtiene los nombres de archivos a donde redirigir la entrada (salida).*

*\* self: comando simple a decidir si está vacío.*

*\* Returns: nombre del archivo a donde redirigir la entrada (salida)*

*\* o NULL si no está redirigida.*

*\* Requires: self!=NULL*

*\*/*

char \* scommand\_to\_string(const scommand *self*);

*/\* Preety printer para hacer debugging/logging.*

*\* Genera una representación del comando simple en un string (aka "serializar")*

*\* self: comando simple a convertir.*

*\* Returns: un string con la representación del comando simple similar*

*\* a lo que se escribe en un shell. El llamador es dueño del string*

*\* resultante.*

*\* Requires: self!=NULL*

*\* Ensures: scommand\_is\_empty(self) ||*

*\* scommand\_get\_redir\_in(self)==NULL || scommand\_get\_redir\_out(self)==NULL ||*

*\* strlen(result)>0*

*\*/*

*/\**

*\* pipeline: tubería de comandos.*

*\* Ejemplo: ls -l \*.c > out < in | wc | grep -i glibc &*

*\* Secuencia de comandos simples que se ejecutarán en un pipeline,*

*\* más un booleano que indica si hay que esperar o continuar.*

*\**

*\* Una vez que un comando entra en el pipeline, la memoria pasa a ser propiedad*

*\* del TAD. El llamador no debe intentar liberar la memoria de los comandos que*

*\* insertó, ni de los comandos devueltos por pipeline\_front().*

*\* pipeline\_to\_string() pide memoria internamente y debe ser liberada*

*\* externamente.*

*\**

*\* Externamente se presenta como una secuencia de comandos simples donde:*

*\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\* front -> | scmd1 | scmd2 | ... | scmdn | <-back*

*\* ------------------------------*

*\*/*

typedef struct pipeline\_s \* pipeline;

pipeline pipeline\_new(void);

*/\**

*\* Nuevo `pipeline', sin comandos simples y establecido para que espere.*

*\* Returns: nuevo pipeline sin comandos simples y que espera.*

*\* Ensures: result != NULL*

*\* && pipeline\_is\_empty(result)*

*\* && pipeline\_get\_wait(result)*

*\*/*

pipeline pipeline\_destroy(pipeline *self*);

*/\**

*\* Destruye `self'.*

*\* self: tubería a a destruir.*

*\* Requires: self != NULL*

*\* Ensures: result == NULL*

*\*/*

*/\* Modificadores \*/*

void pipeline\_push\_back(pipeline *self*, scommand *sc*);

*/\**

*\* Agrega por detrás un comando simple a la secuencia.*

*\* self: pipeline al cual agregarle el comando simple.*

*\* sc: comando simple a agregar. El TAD se apropia del comando.*

*\* Requires: self!=NULL && sc!=NULL*

*\* Ensures: !pipeline\_is\_empty()*

*\*/*

void pipeline\_pop\_front(pipeline *self*);

*/\**

*\* Quita el comando simple de adelante de la secuencia.*

*\* self: pipeline al cual sacarle el comando simple del frente.*

*\* Destruye el comando extraido.*

*\* Requires: self!=NULL && !pipeline\_is\_empty(self)*

*\*/*

void pipeline\_set\_wait(pipeline *self*, const bool *w*);

*/\**

*\* Define si el pipeline tiene que esperar o no.*

*\* self: pipeline que quiere ser establecido en su atributo de espera.*

*\* Requires: self!=NULL*

*\*/*

*/\* Proyectores \*/*

bool pipeline\_is\_empty(const pipeline *self*);

*/\**

*\* Indica si la secuencia de comandos simples tiene longitud 0.*

*\* self: pipeline a decidir si está vacío.*

*\* Returns: ¿Está vacío de comandos simples el pipeline?*

*\* Requires: self!=NULL*

*\*/*

unsigned int pipeline\_length(const pipeline *self*);

*/\**

*\* Da la longitud de la secuencia de comandos simples.*

*\* self: pipeline a medir.*

*\* Returns: largo del pipeline.*

*\* Requires: self!=NULL*

*\* Ensures: (pipeline\_length(self)==0) == pipeline\_is\_empty()*

*\**

*\*/*

scommand pipeline\_front(const pipeline *self*);

*/\**

*\* Devuelve el comando simple de adelante de la secuencia.*

*\* self: pipeline al cual consultar cual es el comando simple del frente.*

*\* Returns: comando simple del frente. El comando devuelto sigue siendo*

*\* propiedad del TAD.*

*\* El resultado no es un "const scommand" ya que el llamador puede*

*\* hacer modificaciones en el comando, siempre y cuando no lo destruya.*

*\* Requires: self!=NULL && !pipeline\_is\_empty(self)*

*\* Ensures: result!=NULL*

*\*/*

bool pipeline\_get\_wait(const pipeline *self*);

*/\**

*\* Consulta si el pipeline tiene que esperar o no.*

*\* self: pipeline a decidir si hay que esperar.*

*\* Returns: ¿Hay que esperar en el pipeline self?*

*\* Requires: self!=NULL*

*\*/*

char \* pipeline\_to\_string(const pipeline *self*);

*/\* Pretty printer para hacer debugging/logging.*

*\* Genera una representación del pipeline en una cadena (aka "serializar").*

*\* self: pipeline a convertir.*

*\* Returns: una cadena con la representación del pipeline similar*

*\* a lo que se escribe en un shell. Debe destruirla el llamador.*

*\* Requires: self!=NULL*

*\* Ensures: pipeline\_is\_empty(self) || pipeline\_get\_wait(self) || strlen(result)>0*

*\*/*